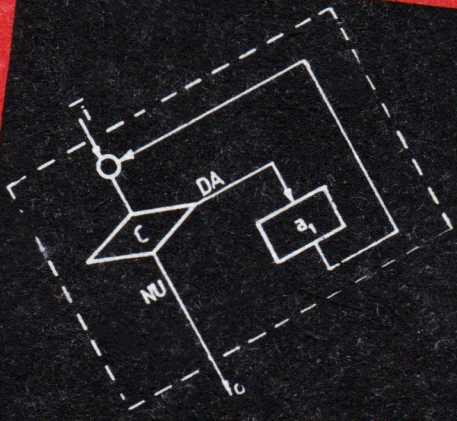
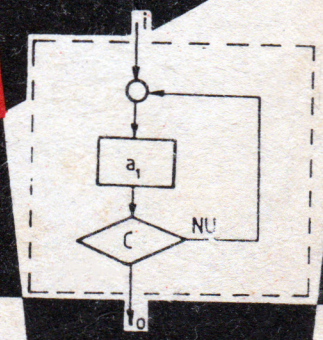
Să învățăm să programăm

Camelia Zaharia
Marian Zaharia

BASIC



EDITURA TEHNICĂ



Marian Zaharia **Camelia Zaharia**

SĂ ÎNVĂŢĂM SĂ PROGRAMĂM

Editura Tehnică

Bucureşti 1992

La capăt trebuie să obţii o transfigurare. Transfigurarea mea e logica.

Problema "problemei" nu este una de logică? Programarea, punerea în ordine a întrebărilor, nu-i revin ei?

Constantin Noica - Jurnal de idei

Încă o carte despre calculatoarele electronice şi utilizarea lor în rezolvarea problemelor pe care şi le-a pus întotdeauna omul!

Nimic mai natural în epoca noastră în care, aşa cum spunea un mare savant: "Ne înnecăm în informaţii, dar sîntem înfometaţi de cunoaştere".

Şi această foame de cunoaştere nu poate fi potolită decît prin astfel de cărţi, cum este cea de faţă, pe care prestigioasa Editură Tehnică o pune generos la dispoziţia unei largi categorii de cititori - elevi, studenţi, profesori, specialişti în informatică.

Tot mai mulţi oameni au nevoie de calculatorul electronic ca de o unealtă de muncă. Alţii îl au ca partener de joacă sau de învăţătură.

În această ultimă ipostază ni-l prezintă şi nouă autorul, cunoscut informatician, cu o bogată experienţă în domeniul programării calculatoarelor. Scopul declarat al cărţii este să ne înveţe cum să formulăm problemele pe care le avem de rezolvat, astfel încît ele să fie înţelese şi rezolvate, mult mai repede şi mai bine, de către calculator.

Este un demers îndrăzneţ, dar experienţa şi talentul pedagogic ale autorului au produs o carte extrem de utilă, în primul rînd viitorilor informaticieni, tinerilor decişi să-şi aleagă o profesie în lumea fascinantă a calculatoarelor electronice.

Conf. dr. Emil Scarlat

INTRODUCERE

"Una din dilemele condiţiei umane este că experienţa şi ştiinta noastră se referă la trecut, în timp ce deciziile le luăm cu privire la viitor. Această dilemă existenţială a generat o practică istorică: omul s-a obişnuit să întîmpine şi să rezolve problemele noi ale dezvoltării, cu experienţa şi învăţămintele trecutului".

Pentru prima oară această practică devine insuficientă; şi pentru ca dilema să poată fi rezolvată, trebuie ca omul să poată învăţa de la viitor în aceeaşi manieră în care a învăţat şi învaţă de la trecut; iar pentru aceasta trebuie, în primul rînd să ştie atît să gîndească logic, sistemic, cît şi să beneficieze de un ajutor substanţial: calculatorul. Iată de ce, şi această carte, pledează pentru a învăţa să folosim - prin propriul nostru filtru - una din ultimele cuceriri ale omului în materie de "tehnică inteligentă", calculatorul electronic.

Existenţa cotidiană ne pune permanenent în faţă probleme din ce în ce mai numeroase şi mai complexe.

A şti CE SĂ FACI - este un prim pas în realizarea oricărui scop propus, a şti CUM SĂ FACI - este nu numai al doilea pas în ordinea firească, dar, mai ales, cheia reuşitei, a îndeplinirii acestuia, echivalentul practic al încărcăturii magice a lui "SESAM DESCHIDE-TE".

Omul, de cînd există, a fost frămîntat de aceste întrebări. Găsind răspunsuri - mai bune sau mai puţin bune - a reuşit să-şi apropie elementele mediului înconjurător, să-şi creeze actualul mediu de viaţă, a evoluat, a creat, dezvoltat şi perfecţionat în jurul său de la propria gîndire, pînă la fabricarea unei gîndiri similare lui, dar artificială, pentru maşinile sale, care să preia din ce în ce mai mult, nu numai sarcinile de rutină ci, mai nou, chiar elemente din activitatea de creaţie - care pînă de curînd i-a aparţinut în întregime.

În această idee, lucrarea de faţă îşi propune să evidenţieze importanţa calităţii răspunsurilor la întrebările CE şi CUM, pentru un nivel concret, dar nu restrîns, acela al abordării aspectelor legate de utilizarea sistemelor automate de calcul cu ajutorul unui limbaj de programare (BASIC), de abordarea, formalizarea şi programarea cît mai eficientă a soluţionării corecte şi rapide a diverselor probleme.

Lucrări în domeniu, care să includă în cuprinsul lor - total sau parţial - asemenea aspecte, s-au scris foarte multe, dar adresate îndeosebi specialiştilor. Ceea ce aduce nou această carte este abordarea conţinutului său într-un mod pedagogic, pe scară graduată de complexitate şi informaţie, într-o viziune integratoare.

Plecînd de la enunţul unei probleme - în general - lucrarea trece prin toate etapele logice ale rezolvării ei - respectiv: teoretizarea, descompunerea pe paşi, conceperea algoritmului, descrierea acestuia - ca apoi să facă introducerea în spaţiul informatizat al rezolvării ei, respectiv alcătuirea programului. Prin parcurgerea acestor etape se urmăreşte formarea şi dezvoltarea unei modalităţi de abordare şi gîndire logică nu numai în rezolvarea problemelor de matematică, dar şi în

soluţionarea oricăror tipuri de probleme cu care ne "ciocnim" în viaţa cotidiană.

De ce s-a ales BASIC, ca limbaj de programare pentru aplicaţiile ce vin să concretizeze aspectele abordate? Pentru că este un limbaj relativ uşor de însuşit atît pentru cei ce nu cunosc programare, cît şi pentru cei familiarizaţi deja cu alte limbaje, cît, în special, datorită vastei sale răspîndiri determinate de apariţia şi dezvoltarea spectaculoasă a calculatoarelor personale (atît de familiare astăzi).

Lucrarea nu urmăreşte prezentarea în mod special a unei anume variante BASIC (o abordare vastă şi complexă a diverselor familii şi generaţii BASIC fiind prezentată în [4]), ci utilizarea acestuia ca instrument în soluţionarea problemelor. Din aceasta cauză ne-am oprit în mod special asupra elementelor fundamentale ale limbajului, de regulă, universal valabile în familiile şi generaţiile sale, referirea la o anume versiune făcîndu-se numai cînd am considerat a fi absolut necesară şi cu specificarea acesteia.

Prin modul de abordare, prin natura problemelor propuse şi soluţionate, prin cele peste 30 de exemple şi 40 de aplicaţii (însoţite de rezultatele execuţiilor efectuate cu date de test) prezentate în capitolul 5, lucrarea are un profund caracter practic-aplicativ .

Lucrarea se adresează, în special, elevilor din ciclurile gimnazial şi liceal, profesorilor care predau noţiuni introductive de programare, precum şi studenţilor şi tuturor acelora care au în propria casă sau la îndemîna un "home-computer" şi vor să-l utilizeze.

Autorii

CUPRINS

1. NOŢIUNI GENERALE 1
1.1 Scurt istoric 1
1.2 Ce este un calculator numeric? 3
- Hardware 4
- Software 6

1.3 Cum rezolvăm o problemă? 7
- Formularea problemei 7
- Fundamentarea teoretică 7
- O primă concluzie 9
- Algoritmul (paşii) rezolvării problemei 9

1.4 Cîteva concluzii 10

2. ALGORITMI. METODE DE REPREZENTARE 11
2.1 Despre algoritmi 11
- Definiţie 11
- Exemple clasice de algoritmi 11
- Conexiunile algoritmilor cu mulţimile valorilor variabilelor de intrare şi de ieşire 14
- Proprietăţile algoritmilor 16

2.2 Descrierea algoritmilor 17
- Schemele logice 17
- Limbajul algoritmic 'pseudocod' 19

2.3 Structuri fundamentale 21
- Structuri liniare 22
- Structuri alternative 23
- Structuri repetitive 25

2.4 Utilizarea variabilelor indexate 30

3. ELEMENTE ALE LIMBAJULUI BASIC 32
3.1 Cîteva precizări 32

3.2 Moduri de operare şi comenzile interpretorului 34
- Iniţializarea interpretorului 34
- Moduri de operare 35
- Salvări, restaurări 36
- Ştergeri, modificări, listări 37
- Execuţii 39
- Reluări, trasări 40
- Revenirea în sistemul de operare 40

3.3 Propoziţii şi fraze (programe) BASIC 40
3.4 Vocabular şi elemente de gramatică 43
- Setul de caractere 43
- Constante 43
- Cuvinte rezervate 43
- Identificatori 44
- Variabile 44
- Operatori 46
- Expresii 47

3.5 Instrucţiuni de intrare / ieşire 48
3.6 Operaţii aritmetice şi logice 54
- Operaţii aritmetice 54
- Operaţii logice 56
- Operaţii cu variabile şir 58

3.7 Funcţii şi subrutine 58
- Funcţii predefinite 59
- Funcţii definite de utilizator 63
- Subrutine BASIC 64
- Subrutine BETA BASIC 67

3.8 Instrucţiuni de control 68
- Instrucţiuni alternative 68
- Instrucţiuni repetitive 70
- Instrucţiuni de salt 72

3.9 Lucrul cu fişiere 76
- Deschiderea şi închiderea fişierelor 76
- Fişiere secvenţiale 77
- Lucrul cu fişiere în acces direct 79

3.10 Înlănţuirea programelor 83
3.11 Elemente de grafică 84
- Fereastra, vizor, scalare, decupare 85
- Instrucţiuni grafice 87

4. SPRE O PROGRAMARE MODULARĂ ŞI STRUCTURATĂ 92

5. APLICAŢII 105
5.1 Numere întregi, baze de numeraţie 106
● Împărţire cu rest a două numere naturale 106
● Împărţirea cu rest a două numere întregi 107
● Determinarea CMMDC al două numere întregi 108
● Determinarea CMMMC al două numere întregi 110
● Determinarea CMMDC al maxim 20 de numere întregi 111
● Determinarea CMMMC al maxim 20 de numere întregi 113
● Generarea de numere prime 115
● Determinarea divizorilor unui număr întreg 118
● Conversia unui număr natural din baza 10 într-o bază mai mică decît 10 .. 119
● Conversia unui număr natural din baza 10 în toate bazele mai mici decît 10 121
● Conversia unui număr binar în baza 10 123
● Descompunerea unui număr întreg în factori primi 125
5.2 Vectori 128
● Determinarea elementului minim / maxim şi al locului său 1ntr-un vector . 128
● Suma elementelor unui vector 130
● Căutarea unei element într-un vector 131
● Determinarea numerelor elementelor pozitive, nule şi negative conţinute într-un vector 132
● Interschimbarea a două elemente ale unui vector 134
● Inversarea ordinii elementelor unui vector 135
● Interschimbarea a k elemente de la începutul unui vector cu cele k elemente de la sfîrşitul său 136
● Ordonarea crescătoare / descrescătoare a elementelor unui vector 137
● Rotirea stînga / dreapta a elementelor unui vector 139
● Deplasarea stînga / dreapta a elementelor unui vector 141
5.3 Mulţimi 143
● Validarea unei mulţimi de numere 143
● Reuniunea a două mulţimi 144
● Intersecţia a două mulţimi 147
● Diferenţa a două mulţimi 148
● Produsul cartezian al două mulţimi de numere 150
5.4 Matrici, determinanţi 151
● Citirea şi listarea unei matrice 151
● Suma elementelor unei matrice 152
● Suma a două matrici 153
● Produsul a două matrici 154

- Transpusa unei matrice 155
- Calculul determinantului unei matrice 156
- Inversarea unei matrici pătrate 159

5.5 Şiruri de caractere 161

- Conversia unui număr natural (32<=N<=26) în caracter 161
- Crearea unui cap de tabel 162
- Căutarea unui subşir într-un şir dat 163

5.6 Fişiere 164

- Actualizarea fişierului secvenţial CLASA 165
- Actualizarea fişierului cu acces direct CLASAD 169

ANEXA A 173

ANEXA B 177

Bibliografie 183

1. NOŢIUNI GENERALE

1.1 Scurt istoric

De-a lungul existenţei sale omul a căutat permanent soluţii pentru a se elibera de efortul fizic, de muncile obositoare sau plictisitoare. Diversificarea continuă a activităţilor sale a determinat şi determină şi acum necesitatea găsirii unor soluţii noi la vechile şi permanentele sale probleme, ceea ce a dus, şi duce, la perfecţionarea continuă a instrumentelor cu care operează.

În confruntarea continuă cu natura şi cu timpul s-a născut de mult ideea inventării unor maşini automate de calculat şi, de ce nu, a unor maşini automate care să-l ajute în gîndire.

Desi idei şi încercări în acest sens datează de prin secolul al XVII-lea (Napier-1617, Pascal-1642) şi mai cu seamă din secolul al XIX-lea (Babbage lansează în 1820 ideea realizării unei maşini de calculat care, pe lîngă efectuarea de operaţii aritmetice să aibă posibilitateta de reţinere a comenzilor necesare soluţionarii problemei, precum şi a rezultatelor, pe care să le furnizeze ulterior), realizarea practică a primelor calculatoare este posibilă abia către jumatatea secolului al XX-lea (descoperirea de către Lee de Forest în anul 1906 a triodei a determinat o puternică dezvoltare a electronicii, bazate pe tuburi electronice, în prima jumătate a acestui secol).

În anul 1945 apare ENIAC (Electronic Numerical Integrator And Calculator) care este considerat a fi primul calculator electronic. Datorită celor circa 20000 de tuburi electronice care îl compuneau, dimensiunile sale erau considerabile.

Apariţia calculatoarelor electronice a produs o puternică revoluţie în modul de abordare şi rezolvare a problemelor, determinînd practic apariţia şi dezvoltarea informaticii - ca ştiinţă a prelucrării automate a datelor - cu puternice implicaţii asupra tuturor domeniilor activităţii umane.

Concomitent cu evoluţiile tehnologice în domeniul dispozitivelor şi circuitelor electronice, calculatorul a evoluat continuu, stăbătînd mai multe generaţii (tabelul 1.1).

Prima generaţie, cuprinzînd calculatoarele electronice din perioada 1946-1956, se caracterizează în principal prin utilizarea ca elemente de bază a tuburilor electronice, fiabilitate scăzută, memorii interne mici, viteze de calcul reduse şi o programare greoaie (în limbaj de asamblare). Datorită dimensiunilor şi dificultăţilor în exploatare au cunoscut o utilizare restrînsă.

Generaţia a doua, cuprinzînd calculatoarele electronice realizate între anii 1957-1963 îşi datorează existenţa dezvoltării şi utilizării tehnologiilor bazate pe tranzistori, ceea ce a determinat o creştere a fiabilităţii şi vitezei de calcul concomitent cu reducerea dimensiunilor.

Apar memoriile bazate pe celule din ferită. Se pun la punct metode de gestiune a resurselor calculatorului; apare necesar, şi este creat, un sistem de programe care să dirijeze funcţionarea optimă a acestuia (sistemul de operare) astfel că începînd cu generaţia a doua, sistemele de calcul se compun din două părţi mari: **hardware** (dispozitivele şi echipamentele componente ale unui sistem) şi **software** (totalitatea programelor cu care este echipat sistemul de calcul). De asemenea apar şi se

dezvoltă limbajele de programare de nivel înalt (universale), relativ independente de calculator (FORTRAN, COBOL etc.). Aria de utilizare a calculatoarelor se măreşte.

GENERAŢII DE CALCULATOARE ELECTRONICE [17]

Generaţie	Perioadă	Principala tehnologie hardware	Alte tehnologii hardware	Limbaje	Alte proprietăţi[1)]
I	1964-1956	Tuburi electronice	Tambur magnetic	De asamblare	MI=2 KO V=10 KIPS
II	1957-1963	Tranzistori	Memorii din ferite	De nivel înalt: FORTRAN, COBOL	MI=32 KO V=200 KIPS
III	1964-1981	Circuite integrate	Memorii semiconductoare, discuri magnetice	De nivel foarte înalt: PASCAL, LISP, limbaje grafice etc.	MI=2 MO V=5 MIPS
IV	1982-1989	Circuite integrate pe scară largă şi foarte largă	Memorii cu bule magnetice, discuri optice	ADA, limbaje orientate obiect	MI=8 MO V=30 MIPS Supercalculatoare
V	după 1989	Circuite integrate pe scară extrem de largă	Arhitecturi paralele	Limbaje concurente, LIMBAJ NATURAL, programare funcţională	Inteligenţă artificială MI>16 MO V>1 GIPS Vedere artificială, tehnologia vorbirii

[1)] MI (memorie internă) 1 MO = 1000 KO
V (viteza de calcul) 1 GIPS = 1000 MIPS = 1000000 KIPS (kiloinstrucţiuni pe secundă)

Tabelul 1.1

Generaţia a treia (1964-1981) este impusă de apariţia circuitelor integrate. Dimensiunile calculatoarelor se reduc considerabil, creşte substanţial dimensiunea memoriei interne (bazată pe semiconductoare), creşte viteza de prelucrare, apare discul magnetic ca suport extern de informaţii de înaltă eficienţă. Apar limbajele de nivel foarte înalt şi limbajele grafice. Apare microprocesorul pe 8 biţi.

Perfecţionarea tehnologiilor, miniaturizarea şi reducerea considerabilă a preţului de cost, au făcut posibilă punerea în practică a dezideratului "un calculator pentru fiecare", astfel încît, anul 1972 (anul realizării microprocesorului 8080) marchează începerea producţiei unor noi tipuri de calculatoare "calculatoarele personale" a căror primă generaţie (tabelul 1.2) se va dezvolta pînă către anul 1980.

Generaţia a patra de calculatoare (1982-1989) se caracterizează prin utilizarea circuitelor integrate pe scară largă şi foarte largă precum şi a microprocesoarelor. Se înregistrează salturi spectaculoase în creşterea capacităţii de stocare pe diferite suporturi externe de informaţii. Preţurile de cost scad considerabil, comparativ cu creşterea performanţelor sistemelor de calcul.

Apariţia în 1980 a microprocesorului pe 16 biţi 8086 determină, în paralel, lansarea generaţiei

a doua de microcalculatoare personale al cărei produs de referinţă este calculatorul IBM-PC. În 1986 este pus la punct microprocesorul 80386. Îi urmează, la scurt timp, microprocesorul 80486 (32 de biţi). Acestea marchează trecerea la a treia generaţie de microcalculatoare personale.

GENERAŢII DE CALCULATOARE PERSONALE

Generaţie	Perioada	Microprocesor	Alte caracteristici
I	1972-1980	8 biţi: 8080, Z-80, 8085, MOS6502, etc.	MI=64 KO Sistem de operare CP/M
II	1981-1986	16 biţi: 8086, 8088, 80286, etc.	MI=640-1000 KO Hard disk de zeci de MO, facilităţi grafice, sistem de operare MS-DOS
III	după 1986	16 şi 32 biţi: 80386, 80486, etc.	MI>1 MO, hard disk de sute de MO, facilităţi grafice deosebite, posibilităţi de lucru în reţea

Tabelul 1.2

După numai circa 45 de ani de evoluţie, transformările suferite de calculatorul electronic din punct de vedere al realizării tehnologice sînt nu numai fundamentale dar şi spectaculoase astfel că, astăzi, calculatorul a devenit prezent practic, în toate domeniile de activitate socială sau personală, a ajuns un instrument de lucru din ce în ce mai necesar.

Care ar fi perspectivele? Încă din 1981 Japonia lansează proiectul calculatorului de **generaţia a cincea**. Această generaţie (a cărei apariţie se întrevede a avea loc după 1990) ar urma să se deosebească de generaţiile anterioare (în speţă de generaţia a patra) în principal, nu atît printr-o nouă tehnologie de construcţie (hardware), ci printr-o nouă concepţie software, aceea a inteligenţei artificiale.

În concepţia specialiştilor japonezi calculatoarele din această generaţie vor deveni sisteme de procesare a informaţiei de cunoaştere. Se presupune că această generaţie va definitiva orientarea informaţională a societăţii umane cu implicaţii majore în evoluţia acesteia.

1.2 Ce este un calculator numeric?

După cum îl înfăţişează şi numele, în general, *un calculator este un sistem fizic care prelucrează datele într-o formă prestabilită şi furnizează rezultatele într-o formă accesibilă utilizatorului sau altui dispozitiv cu care este conectat.*

Din mulţimea de tipuri de calculatoare, calculatoarele numerice sînt acelea care prelucrează informaţii codificate numeric efectuînd operaţii aritmetice, funcţii logice şi transferuri de informaţii.

Aşa cum am văzut, începînd cu generaţia a doua, calculatorul electronic este alcătuit din două componente de bază:

- **hardware** (totalitatea dispozitivelor şi echipamentelor fizice) calculatorul propriu-zis;
- **software** (totalitatea programelor cu care este "echipat" acesta).

● Hardware

Din punct de vedere **hardware** un calculator electronic numeric trebuie să dispună în principal de următoarele componente:

- memorie internă (MI);
- unitate centrală (UC);
- dispozitive periferice (P):
  - de introducere a datelor;
  - de afişare a rezultatelor;
  - eventual, alte dispozitive periferice.

Cum ne interesează în special microcalculatoarele, ne oprim puţin asupra principalelor componente ale acestora (figura 1.1).

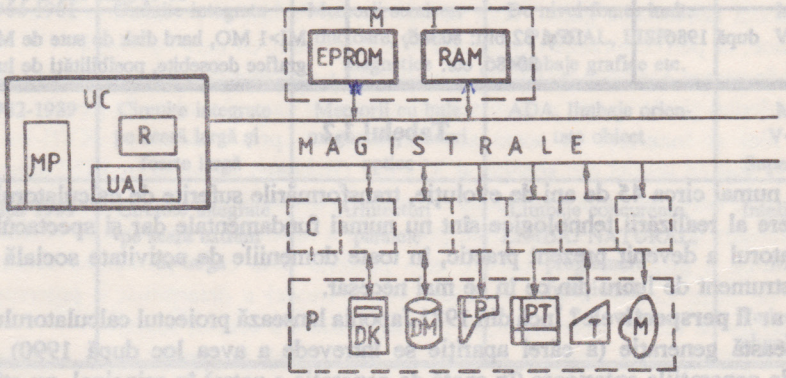


Fig. 1.1

Memoria internă (MI) este un dispozitiv alcătuit din 'locaţii' succesive adresabile, avînd rolul de a memora principalele programe care asigură gestiunea resurselor calculatorului, a programului utilizatorului, precum şi a datelor cu care se lucrează. În general, memoria internă este împărţită în două părţi distincte: memoria EPROM (conţine informaţii privitoare la diferitele caracteristici fizice ale maşinii şi programul "încărcător", program ce preia controlul la alimentarea cu tensiune a calculatorului), avînd proprietatea de a-şi păstra conţinutul şi după decuplarea calculatorului, dar care permite numai citirea informaţiilor stocate, şi memoria RAM care permite operaţii de citire / scriere şi în care sînt reţinute temporar programe şi date. Memoria RAM îşi pierde conţinutul la decuplarea alimentării calculatorului.

Unitatea centrală (UC) este un dispozitiv care asigură dirijarea funcţionării normale atît a calculatorului în ansamblu cît şi a principalelor sale componente. Unitatea centrală ia decizii privind modul şi ordinea de execuţie a diferitelor operaţii în interiorul calculatorului şi semnalează apariţia unor condiţii de eroare sau funcţionarea anormală a acestuia.

Componenta fundamentală a unităţii centrale o constituie microprocesorul (μP), el fiind de fapt elementul care, prin sistemul de întreruperi, coordonează funcţionarea tuturor componentelor calculatorului. După cum am văzut din tabelul 1.2, microprocesorul, prin caracteristicile sale, este un

element definitor al generaţiilor de microcalculatoare şi, implicit, al performanţelor unui calculator. O importanţă aparte, prin rolul său în funcţionarea unităţii centrale, o are unitatea aritmetică şi logică (UAL), unitate ce asigură execuţia efectivă a operaţiilor aritmetice şi logice.

De asemenea, unitatea centrală, mai conţine o memorie proprie (registre - R) precum şi o serie de alte circuite menite să asigure sau să îmbunătăţească performanţele calculatorului.

Pentru a executa o instrucţiune (de exemplu adunarea a două numere C = A + B) unitatea centrală execută următoarele operaţii:

- preia din memoria internă instrucţiunea (C = A+B) şi o decodifică;
- calculează adresele operanzilor, îi identifică (A,B) şi efectuează deplasările necesare (aduce operandul A în memoria proprie);
- cu ajutorul UAL efectuază operaţia (A + B) cu păstrarea rezultatului;
- depune rezultatul în memoria internă la locaţia corespunzătoare (C);
- semnalează sfîrşitul şi corectitudinea executării operaţiei.

Referitor la funcţionarea unităţii centrale trebuie subliniat faptul că ea execută operaţiile pas cu pas (pe rînd) la un moment dat executînd o singură operaţie.

Dispozitivele periferice de introducere a datelor sînt multiple. În cazul calculatoarelor personale acest dispozitiv îl constituie, de obicei, tastatura (T).

Dispozitivele de afişare a datelor alcătuiesc o categorie complexă şi variată de dispozitive. Din aceasta fac parte, în primul rînd monitorul (M) - banalul televizor - care poate fi color sau monocrom, dispozitiv utilizat şi pentru conversaţia cu calculatorul, apoi imprimanta (alfanumerică sau grafică), plotterul, etc.

Pe lîngă aceste dispozitive periferice, o importanţă deosebită o au suporturile de memorie externă, fără de care, în condiţiile actuale este greu de conceput utilizarea eficientă a unui calculator. Preţurile relativ scăzute în ultima vreme ale acestor componente le fac tot mai acesibile utilizatorului neprofesionist, avantajele utilizării lor fiind evidente. În această categorie amintim numai, benzile magnetice (BM), discurile magnetice (DM) (cu capacităţi de stocare de ordinul sutelor de megaocteţi), precum şi familiarele dischete (DK) (şi ele cu capacităţi de ordinul megaocteţilor).

Atît memoria internă cît şi dispozitivele periferice sînt conectate la unitatea centrală prin magistrale - prin care se asigură atît transferul datelor şi instrucţiunilor cît şi controlul funcţionării acestora. Conectarea dispozitivelor periferice la magistrale este efectuată cu ajutorul unor dispozitive (cuploare, interfeţe) specifice fiecăruia.

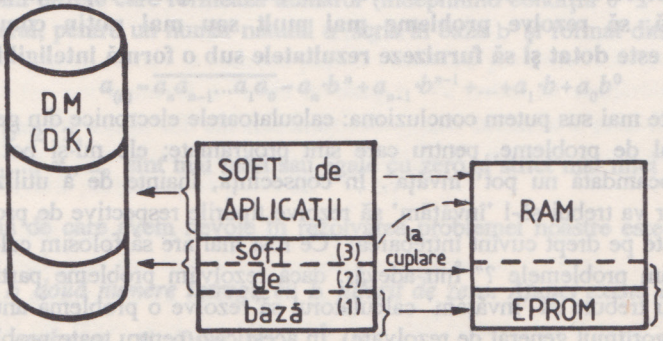


Fig. 1.2

● Software

Cea de a doua componentă de bază a unui calculator numeric este cunoscută sub numele de software şi, la rîndul său, este constituită din două părţi mari (figura 1.2):

- software-ul de bază;
- software-ul de aplicaţii.

Software-ul de bază este alcătuit din totalitatea programelor fără de care utilizarea calculatorului nu este posibilă. O parte din software-ul de bază (1) este depus în memoria EPROM. La cuplarea calculatorului în memoria internă sînt încărcate automat de pe disc şi alte programe absolut necesare dirijării funcţionării calculatorului (2), programe care împreună cu cele aflate în memoria EPROM alcătuiesc 'nucleul sistemului de operare', nucleu existent în memoria internă a calculatorului pe toată durata funcţionării lui. Restul componentelor (3) sînt încărcate în memorie numai dacă sînt solicitate de sistem sau utilizator şi numai pe durata utilizării lor (din această categorie fac parte compilatoarele, diferitele utilitare etc.).

Observaţie. În cazul microcalculatoarelor care nu dispun de unităţi de disc toate componentele software legate de funcţionarea calculatorului precum şi un minim pentru exploatarea acestuia se află permanent în memoria EPROM. În cazul acestor calculatoare nu putem vorbi de "sistem de operare" în înţelesul deplin al cuvîntului.

Software-ul de aplicaţii este constituit din mulţimea programelor destinate rezolvării diferitelor tipuri de probleme.

În această categorie intră şi programele create de utilizator; aceste programe sînt lansate în execuţie, de regulă de utilizator şi sînt prezente în memorie numai pe durata execuţiei.

Observaţie. În cazul microcalculatoarelor care nu dispun de discuri magnetice (fixe sau dischete) pentru încărcarea/salvarea programelor este necesară existenţa cel puţin a unei unităţi de bandă magnetică (de regulă casetofon), comanda pentru efectuarea operaţiei (salvare/restaurare) fiind dată de utilizatotr; în caz contrar programul 'se pierde' la decuplarea calculatorului.

Calculatorul electronic numeric este deci o maşină complexă capabilă să primească date sub formă numerică, să rezolve probleme mai mult sau mai puţin complicate conform programelor cu care este dotat şi să furnizeze rezultatele sub o formă inteligibilă utilizatorului.

Din cele arătate mai sus putem concluziona: calculatoarele elecronice din generaţiile I-IV pot soluţiona numai genul de probleme, pentru care sînt programate; ele nu-şi pot construi singure algoritmi şi deci, deocamdată nu pot 'învăţa'. În consecinţă, înainte de a utiliza calculatorul la rezolvarea problemelor va trebui să-l 'învăţăm' să rezolve tipurile respective de probleme.

S-ar putea naşte pe drept cuvînt întrebarea: "Ce rost mai are să folosim calculatorul dacă tot noi trebuie să rezolvăm problemele ?" Într-adevăr, dacă rezolvăm probleme particulare întrebarea este justificată. Dar nu trebuie să 'învăţăm' calculatorul să rezolve o problemă anume ci, un tip de probleme (să-i dăm algoritmul general de rezolvare). În acest caz, pentru toate problemele de acel tip dîndu-i numai datele specifice (date de intrare) calculatorul va determina, urmărind algoritmul general,

răspunsul căutat. De data aceasta utilizarea calculatorului este eficientă deoarece soluţiile problemei se obţin fără a mai participa efectiv la efectuarea calculelor.

Calculatorul este deci, o maşină capabilă să rezolve orice problemă cu condiţia să fi fost 'învăţat' cum să o facă.

În consecinţă, înainte de toate, cum rezolvăm o problemă?

1.3 Cum rezolvăm o problemă?

În general problemele pe care dorim să le rezolvăm cu calculatorul sînt teoretic cunoscute dar ele necesită fie un volum mare de calcule, fie operaţii stereotipe, plictisitoare.

În aceste cazuri comod este să introducem datele problemei în calculator şi să obţinem rezultate. Am văzut însă că, pentru a putea determina soluţiile unei probleme cu ajutorul calculatorului, este necesar ca acesta 'să cunoască' pas cu pas întreaga succesiune de operaţii ce trebuie efectuate pentru rezolvarea acesteia.

Să rezolvăm o problemă simplă (de nivelul clasei a V-a): trecerea unui număr natural din baza 10 într-o bază b, cu b mai mare sau egal cu 2 şi mai mic decît 10.

Tocmai pentru că este o problemă banală să vedem cum o rezolvăm sau, mai bine zis, cum o pregătim pentru a putea fi rezolvată pe calculator.

● Formularea problemei

Să se treacă un număr natural a din baza 10 în baza b ($b \in N$ şi $2 \le b < 10$)

● Fundamentarea teoretică

Un număr natural, de exemplu 157, spunem că este scris în baza 10 dacă:

$$157_{(10)} = 1 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0$$

unde 1, 5 şi 7 sînt cifrele care formează numărul (îndeplinind condiţia $0 \le 1, 5, 7 \le 9$) iar 10 este baza. În general, pentru un număr natural a scris în baza b şi format din n+1 cifre avem:

$$a_{(b)} = \overline{a_n a_{n-1} \ldots a_1 a_0} = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b + a_0 b^0$$

unde: $b \in N$ şi $b \ge 2$,
iar cifrele $a_n, a_{n-1}, \ldots, a_1, a_0$ sînt mai mari sau egale cu zero şi strict mai mici decît b.

O teoremă de care avem nevoie în rezolvarea problemei noastre este teorema împărţirii cu rest:

Fie a şi b două numere întregi, cu b diferit de zero. Atunci există două numere întregi q şi r, astfel încît: $a = bq + r$ *şi* $0 \le r < |b|$.

Numerele q şi r determinate în aceste condiţii sînt unice.

Cu alte cuvinte, împărţind un număr întreg a (deîmpărţitul) la un număr întreg b (împărţitor) obţinem un cît q şi un rest r. Restul este totdeauna mai mic decît împărţitorul, luat în valoare absolută, şi mai mare sau egal cu 0. Dacă r (restul) este egal cu zero, atunci spunem că b divide a sau că b este divizor al lui a.

Exemplul 1.1

Fie $a = 125_{(10)}$; să se determine numărul $c_{(2)} = a_{(10)}$.

Aplicînd teorema împărţirii cu rest şi parcurgînd schema din figura 1.3 rezultă:

$$125_{(10)} = 1111101_{(2)}.$$

Verificare: utilizînd modul de scriere al numerelor în baza 2 rezultă:

$$1111101_{(2)} = 1\cdot2^6+1\cdot2^5+1\cdot2^4+1\cdot2^3+1\cdot2^2+0\cdot2^1+1\cdot2^0 = 125_{(10)}$$

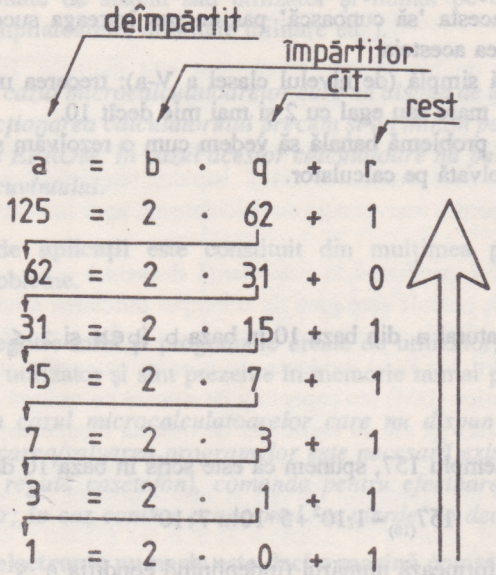


Fig. 1.3

Exemplul 1.2

Să convertim în baza 8 numărul $342_{(10)}$. Folosind schema din figura 1.4 rezultă:

$$342_{(10)} = 526_{(8)}$$

Verificare:

$$526_{(8)} = 5\cdot8^2+2\cdot8^1+6\cdot8^0$$

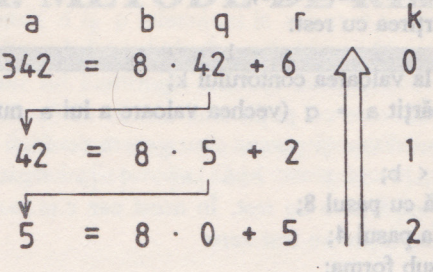


Fig.1.4

Privind comparativ cele două exemple observăm că am folosit aceeaşi metodă, şi anume, am împărţit (utilizînd teorema împărţirii cu rest) numărul dat (125 respectiv 342) scris în baza zece la noua bază (2 respectiv 8) obţinînd un cît şi un rest. Am reţinut restul iar cîtul a devenit deîmpărţit şi am efectuat o nouă împărţire. Pînă cînd? Pînă cînd cîtul a ajuns mai mic decît împărţitorul şi deci, în ultima împărţire, a devenit el însuşi rest. Numărul căutat se formează din cifrele rezultate ca rest, luate în ordine inversă obţinerii lor.

Examinaţi ultimul paragraf. Ce constataţi ?

● O primă concluzie

Modul de rezolvare nu depinde nici de numărul dat, nici de baza aleasă (respectînd însă condiţiile impuse). Am alcătuit deci o metodă generală care, transpusă pe calculator, ne permite rezolvarea problemei pentru orice pereche de numere naturale `(a,b)` cu `b` mai mare sau egal cu 2 şi strict mai mic decît 10.

După cum am văzut însă în capitolul 1.2 deocamdată calculatorul rezolvă problemele numai 'pas cu pas'. **Care ar fi paşii rezolvării problemei propuse ?**

● Algoritmul (paşii) rezolvării problemei

Înainte de a trece efectiv la scrierea paşilor rezolvării problemei, să facem o precizare: pentru a putea şti cîte cifre are numărul `a` scris în baza `b` va trebui să numărăm de cîte ori efectuăm împărţirea cu rest; pentru aceasta vom folosi un `contor k` cu valoare iniţială egală cu zero în care, la fiecare trecere (de fiecare dată cînd executăm împărţirea), adunăm o unitate.

Paşii rezolvării problemei ar putea fi:

- **pasul 1**: se stabilesc valorile numerelor `a` şi `b`;
- **pasul 2**: se verifică dacă sînt îndeplinite condiţiile impuse: `(a,b∈N` şi `2 ≤ b ≤ 9)`;
  - dacă DA continuăm cu pasul 3;
  - dacă NU cel puţin unul din numerele date este eronat ales şi rezolvarea problemei se întrerupe (se sare la pasul 9 sau, eventual, la pasul 1 pentru a corecta eroarea);

- **pasul 3**: se atribuie contorului k valoarea 0;
- **pasul 4**: se efectuează împărţirea cu rest:

$$a = b\,q + r_k$$

- **pasul 5**: se adună o unitate la valoarea contorului k;
- **pasul 6**: cîtul devine deîmpărţit a = q (vechea valoare a lui a nu ne mai interesează), iar restul r_k se reţine;
- **pasul 7**: se verifică dacă a < b;
  - dacă DA se continuă cu pasul 8;
  - dacă NU se revine la pasul 4;
- **pasul 8**: se scrie rezultatul sub forma:

$$\overline{r_k r_{k-1} \ldots r_1 r_0}$$

- **pasul 9**: stop.

După cum vedem modul de pregătire a rezolvării unei probleme pentru a putea fi transpusă pe calculator diferă întrucîtva de modul în care se rezolvă în general problemele. Aceasta se datorează caracterului de generalitate pe care trebuie să-l aibă metoda de rezolvare pe care dorim să o utilizăm.

1.4 Cîteva concluzii

Cel puţin pînă în prezent calculatoarele pe care le avem la îndemînă nu sînt dotate cu **inteligenţa de a învăţa**. Ele sînt nişte maşini care pot soluţiona rapid şi corect o gamă foarte largă de probleme cu condiţia să fi fost **învăţate** să o facă.

Transpunerea pe calculator a modului de rezolvare a unei probleme presupune ca primă condiţie **cunoaşterea teoretică profundă a problemei şi determinarea riguroasă a unei succesiuni de operaţii (paşi) care (în condiţiile în care se îndeplinesc restricţiile precizate) să ducă în mod univoc la determinarea soluţiei (soluţiilor) acesteia.**

Calculatorul execută automat (pas cu pas) operaţiile precizate fără a participa în vreun fel la corectarea eventuală a acestora sau la soluţionarea situaţiilor ambigue. Din această cauză **în precizarea succesiunii operaţiilor pentru rezolvarea unei probleme trebuie analizate şi precizate operaţiile de executat în toate cazurile ce pot apare.**

Transpunerea pe calculator a modurilor de rezolvare ale unor tipuri de probleme nu este o treabă prea simplă dar nici foarte complicată. Ea presupune, însă, pe lîngă cunoaşterea teoretică aprofundată a problemei (problemelor) şi **unele cunoştinţe privind întocmirea şi reprezentarea algoritmilor, precum şi cel puţin a unui limbaj de programare.**

2. ALGORITMI. METODE DE REPREZENTARE

2.1 Despre algoritmi

● Definiţie

În rezolvarea diferitelor probleme (nu numai de matematică), funcţie de complexitatea acestora, trebuie, de obicei, ca pentru a obţine soluţia căutată să executăm, pornind de la anumite date, o succesiune de operaţii aritmetice şi/sau logice. Ştim de asemenea din experienţă că aceste operaţii nu se fac la întîmplare, ci într-o ordine determinată de natura problemei şi de unele rezultate intermediare obţinute. Pentru a putea executa această succesiune de operaţii în mod automat (cu ajutorul calculatorului), deci fără aportul creator al omului, trebuie (aşa cum am văzut în capitolul 1.3) ca ele să fie valabile pentru toate problemele de acelaşi tip, astfel încît schimbînd numai datele iniţiale, soluţiile să fie determine în mod automat.

De asemenea, am mai văzut că fiecare operaţie trebuie să aibă un mod cunoscut de efectuare, numărul lor trebuie să fie finit, iar parcurgerea întregii succesiuni să se încadreze într-un interval de timp rezonabil.

Odată stabilite operaţiile şi ordinea în care urmează a fi executate (în condiţiile precizate mai sus) spunem că am alcătuit un algoritm de rezolvare a problemei respective.

Deci, am putea concluziona că, **un algoritm este o mulţime finită de operaţii cunoscute care se execută într-o ordine stabilită, astfel încît, pornind de la un set de date (datele problemei) ce îndeplinesc anumite condiţii, obţinem, într-un interval de timp finit, un set de valori (soluţiile problemei).**

● Exemple clasice de algoritmi

Să ne oprim asupra a doi algoritmi îndeobşte cunoscuţi: *algoritmul lui Euclid* de determinare a celui mai mare divizor comun a două numere întregi şi *algoritmul lui Eratostene* de determinare a numerelor prime cuprinse în intervalul [2, *n*], unde *n* este un număr natural mai mare sau egal cu doi, algoritm cunoscut sub numele de *ciurul lui Eratostene*.

Exemplul 2.1.

Algoritmul lui Euclid: pentru a obţine c.m.m.d.c. a două numere întregi `a` şi `b` cu `b` diferit de zero împărţim pe `a` cu `b`; dacă restul împărţirii (r_1) este zero atunci `b` este c.m.m.d.c.; dacă nu, `împărţim pe b` cu restul împărţirii anterioare (r_1) şi obţinem restul r_2; apoi împărţim pe r_1 cu r_2 şi obţinem r_3 şi aşa mai departe. Ultimul rest nenul este c.m.m.d.c. al numerelor date (`a` şi `b`).

Înainte de a explicita algoritmul se impun două precizări:

1. Deoarece dacă d este c.m.m.d.c. al numerelor a şi b atunci şi -d este c.m.m.d.c.al lui a şi b şi cum c.m.m.d.c. al numerelor a şi b nu depinde nici de ordinea, nici de semnul lor, atunci, pentru a afla c.m.m.d.c. este suficient să aplicăm algoritmul lui Euclid pentru |a| şi |b|, adică pentru două numere naturale din care cel puţin unul trebuie să fie diferit de zero.

2. Din motive de generalitate şi pentru a nu genera confuzii în algoritm se vor utiliza două varabile d = max(|a|, |b|) şi i = min(|a|, |b|) reprezentînd deîmpărţitul, respectiv împărţitorul cunoscuţi de la împărţirea cu rest. În acest caz c.m.m.d.c. al numerelor a şi b va fi numărul i obţinut la prima împărţire cu rest zero.

Transpus pe paşi (operaţii), algoritmul este:

- **pasul 1**: se aleg numerele întregi a şi b;
- **pasul 2**: |a| > |b| ?
  - dacă DA, atunci d = |a| şi i = |b|;
  - dacă NU, atunci d = |b| şi i = |a|;
- **pasul 3**: i > 0 ?
  - dacă DA, atunci se continuă cu pasul 4;
  - dacă NU, atunci algoritmul nu se poate aplica şi se sare la pasul 10; (în acest punct putem alcătui algoritmul şi în altă variantă, cu salt la pasul 1 pentru o eventuală reluare cu alte valori atribuite variabilelor a şi b)
- **pasul 4**: se efectuează împărţirea:
  d = i q + r;
- **pasul 5**: r = 0 ?
  - dacă DA, atunci se sare la pasul 9;
  - dacă NU, atunci se continuă cu pasul 6;
- **pasul 6**: d = i (împărţitorul devine deîmpărţit);
- **pasul 7**: i = r (restul devine împărţitor);
- **pasul 8**: se sare la pasul 4;
- **pasul 9**: i este c.m.m.d.c. al lui a şi b;
- **pasul 10**: stop.

Verificăm cu un exemplu numeric:

- **pasul 1**: a = 21 b = 14;
- **pasul 2**: |21| > |14| ?
  - DA => d = 21 i = 14;
- **pasul 3**: i > 0 (14 > 0) ?
  - DA => se trece la pasul 4;
- **pasul 4**: d = i q + r (21 = 14 1 + 7) => r = 7;
- **pasul 5**: r = 0 ?
  - NU => se trece la pasul 6;
- **pasul 6**: d = 14;
- **pasul 7**: i = 7;
- **pasul 8**: se sare la pasul 4;
- **pasul 4**: d = i q + r (14 = 7 2 + 0) => r = 0;

- **pasul 5**: r = 0 ?
  - DA => se sare la pasul 9;
- **pasul 9**: 7 = (21,14) 7 este c.m.m.d.c. al numerelor 21 şi 14
- **pasul 10**: stop.

Rezultatul este corect. Înlocuind pe a şi b cu alte valori (respectînd însă, bineînţeles, condiţiile impuse) se vor obţine soluţii corecte. Deci transpunerea pe paşi a algoritmului a fost efectuată corect.

Exemplul 2.2.

Ciurul lui Eratostene permite, aşa cum am arătat obţinerea tuturor numerelor prime mai mici sau egale cu un număr natural $n \geq 2$ dat. Metoda constă în următoarele: fiind dat şirul numerelor naturale mai mari sau egale cu 2 şi mai mici sau egale cu n, se porneşte de la primul număr din şir (numărul 2) care este număr prim şi se înlătură din şir toţi multiplii acestuia mai mici sau egali cu n; se trece apoi la următorul număr rămas în şir (numărul 3) şi se înlătură toţi multipli acestuia care mai sînt în şir, şi aşa mai departe pînă la cel mai mare număr prim mai mic sau egal cu $\sqrt{n}$. Numerele rămase în şir constituie mulţimea numerelor prime căutate.

Fie p_k numărul prim căruia îi determinăm la cea de-a k parcurgere multiplii; să schiţăm paşii algoritmului :

- **pasul 1**: se alege numărul n ($n \in N$ şi $n \geq 2$);
- **pasul 2**: se generează şirul numerelor naturale din intervalul [2,n];
- **pasul 3**: k = 1 (sîntem la primul număr prim);
- **pasul 4**: se atribuie lui p_k valoarea celui de al k-lea număr din şir;
- **pasul 5**: se verifică dacă $p < \sqrt{n}$
  - dacă DA, atunci se trece la pasul 6;
  - dacă NU, atunci se trece la pasul 9;
- **pasul 6**: se determină toţi multiplii lui p_k mai mici sau egali cu n şi se înlătură din şir;
- **pasul 7**: se adună 1 la valoarea curentă a lui k;
- **pasul 8**: se sare la pasul 4;
- **pasul 9**: se afişează numerele rămase în şir; acestea formează mulţimea numerelor prime căutate;
- **pasul 10**: stop.

Verificăm cele arătate pentru n = 26:

- **pasul 1**: n = 26;
- **pasul 2**: şirul este:

2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	

- **pasul 3**: k = 1;
- **pasul 4**: $p_k = 2$;
- **pasul 5**: $p_k \leq \sqrt{n}$ $(2 \leq \sqrt{26})$?
  - DA => se trece la pasul 6;

- **pasul 6**: se determină şi se înlătură din şir multiplii lui 2:

  2 3 5 7 9 11 13 15 17 19 21 23 25 ;

- **pasul 7**: k = 2;
- **pasul 8**: se sare la pasul 4;
- **pasul 4**: $p_k = 3$;
- **pasul 5**: $p_k \leq \sqrt{n}$ $(3 \leq \sqrt{26})$?
  - DA => se trece la pasul 6;
- **pasul 6**: se determină şi se înlătură din şirul rămas multiplii lui 3:

  2 3 5 7 11 13 17 19 23 25 ;

- **pasul 7**: k = 3;
- **pasul 8**: se sare la pasul 4;
- **pasul 4**: $p_k = 5$ (5 este al treilea număr rămas în şir);
- **pasul 5**: $p_k \leq \sqrt{n}$ $(5 \leq \sqrt{26})$?
  - DA => se trece la pasul 6;
- **pasul 6**: se determină şi se elimină din şirul rămas multiplii lui 5:

  2 3 5 7 11 13 17 19 23 ;

- **pasul 7**: k = 4;
- **pasul 8**: se sare la pasul 4;
- **pasul 4**: $p_k = 7$;
- **pasul 5**: $p_k \leq \sqrt{n}$ $(7 \leq \sqrt{26})$?
  - NU => se sare la pasul 9;
- **pasul 9**: numerele prime aparţinînd intervalului [2,26] sînt:

  2 3 5 7 11 13 17 19 23

- **pasul 10**: stop.

Soluţia este corectă.

● Conexiunile algoritmilor cu mulţimile valorilor variabilelor de intrare şi de ieşire

Dacă analizăm cele două exemple prezentate, privite de data aceasta din exterior (considerînd fiecare algoritm ca o "cutie", ca un aparat) observăm că obţinerea uneia sau altei soluţii depinde de datele de intrare şi nu de ce se întîmplă în interiorul algoritmului respectiv. Aceasta ne îndeamnă să vedem algoritmul şi ca un **sistem închis** (figura 2.1) **care are două legături cu exteriorul: una de intrare (i) şi una de ieşire (o).**

Pe de altă parte, aşa cum am văzut, algoritmii sînt generali şi reprezintă o succesiune mai simplă sau mai complicată de relaţii care, de fapt, nu sînt altceva decît legături între mulţimea valorilor variabilelor de intrare şi mulţimea soluţiilor. Notînd cu I mulţimea valorilor variabilelor de intrare şi cu O mulţimea soluţiilor putem defini algoritmul ca o funcţie

$$f : I \dashrightarrow O .$$

Sigur, această funcţie poate fi banală (**dacă ne-am propune, de exemplu, să întocmim un**

algoritm care să ne furnizeze distanţa parcursă de un mobil în mişcare rectilinie uniformă am găsi că f nu este altceva decît:

$$f(t) = s = v \cdot t$$

adică o funcţie de gradul întîi) sau ca în majoritatea cazurilor mai mult sau mai puţin complicată (vezi exemplele prezentate anterior).

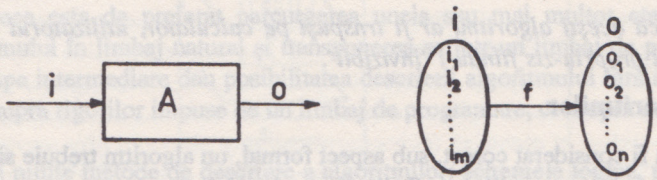


Fig. 2.1

Fără a mai complica lucrurile şi fără a intra în amănunte trebuie subliniat totuşi şi acest mod de interpretare a algoritmului, tocmai pentru a scoate în evidenţă rolul, locul şi interdependenţa dintre algoritmul propriu-zis şi cele două mulţimi de valori ale variabilelor cu care operează (intrările şi ieşirile).

Interacţiunea între algoritmul propriu-zis şi mulţimea valorilor variabilelor de intrare are loc, de regulă, în primul (primii) pas (paşi) ai săi iar legătură cu mulţimea valorilor soluţiilor se efectează în unul (sau mai mulţi) din ultimii săi paşi.

Observaţie. *La conectarea algoritmului cu mulţimea valorilor variabilelor de intrare s-a utilizat noţiunea de 'interacţiune', deoarece într-un algoritm, înainte de a începe parcurgerea operaţiilor propriu-zise, este necesară verificarea condiţiilor ce trebuie îndeplinite de variabilele de intrare (dacă valorile lor aparţin mulţimii valorilor admise); în cazul în care valorile acestora nu corespund condiţiilor impuse, algoritmul trebuie să le 'refuze', iar parcurgerea în continuare a acestuia să fie abandonată.*

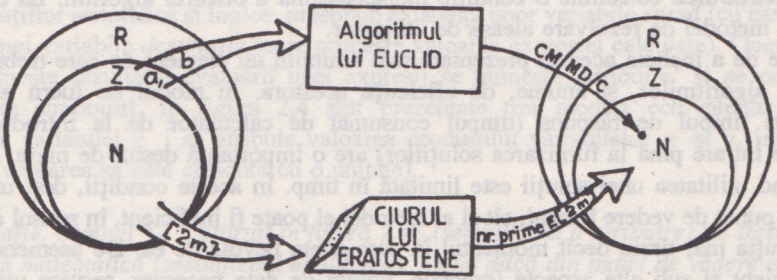


Fig. 2.2

În figura 2.2 sînt prezentate sugestiv locul şi relaţiile dintre cei doi algoritmi prezentaţi (algoritmul lui Euclid şi ciurul lui Eratostene) şi mulţimile valorilor variabilelor cu care acestea operează. Dacă reanalizăm algoritmul lui Euclid observăm că interacţiunea dintre acesta şi mulţimea valorilor variabilelor de intrare (`a` şi `b`) are loc în paşii 1-3 iar soluţia este furnizată la pasul 9 (o constituind variabila de ieşire a cărei valoare aparţine mulţimii soluţiilor problemei, adică `o=f(a,b)`). În mod analog stau lucrurile şi în cazul variantei de algoritm prezentată pentru ciurul lui Eratostene.

Remarcă. Dacă aceşti algoritmi ar fi trnspuşi pe calculator, utilizatorul nu ar sesiza decît aceşti paşi, algoritmul propriu-zis fiindu-i 'invizibil'.

● Proprietăţile algoritmilor

Pentru a putea fi considerat corect, sub aspect formal, un algoritm trebuie să îndeplinească cel puţin trei condiţii esenţiale: **generalitatea, finitudinea şi unicitatea.**

Generalitatea este o proprietate asupra căreia am insistat şi pînă acum (mai ales în capitolul 1.3). Această proprietate, de a fi valabil pentru un gen (o clasă) de probleme, justifică însăşi sensul (scopul) pentru care a fost întocmit.

Finitudinea este proprietatea unui algoritm de a permite obţinerea soluţiei (soluţiilor) problemei după parcurgerea unui număr finit de operaţii. Dacă dintr-un motiv sau altul, (de obicei dintr-o eroare în logica întocmirii) în parcurgerea unui algoritm o operaţie (sau un grup de operaţii) se execută la infinit spunem că s-a intrat într-o "ciclare". Dintr-o asemenea stare algoritmul nu mai poate fi întrerupt decît printr-o intervenţie exterioară.

Unicitatea este acea proprietate a unui algoritm, conform căreia ori de cîte ori se porneşte de la acelaşi set de valori al variabilelor de intrare se obţin aceleaşi soluţii. De asemenea această proprietate presupune existenţa unui mod cunoscut de rezolvare a fiecărei operaţii (acţiuni) cuprinse în algoritm. Aşa cum am mai arătat şi în capitolul 1.3, pentru a conferi unui algoritm această proprietate este necesară abordarea sub toate aspectele a problemei căreia îi întocmim algoritmul, astfel încît să existe variante de urmat pentru toate situaţiile posibile, să fie înlăturate toate ambiguităţile.

Aceste trei proprietăţi caracterizează corectitudinea unui algoritm numai sub aspect formal. Dacă un algoritm are aceste trei proprietăţi nu înseamnă însă că el furnizează şi soluţii corecte, că este corect şi din punct de vedere al problemei pentru care este întocmit.

Corectitudinea constituie o condiţie indispensabilă a oricărui algoritm. Ea depinde însă de corectitudinea metodei de rezolvare aleasă de utilizator.

Înainte de a încheia această prezentare mai amintim un element de care trebuie ţinut seama în întocmirea algoritmilor, şi anume, de **eficienţa** acestora. În modul de lucru actual (în speţă conversaţional) timpul de răspuns (timpul consumat de calculator de la introducerea valorilor variabilelor de intrare pînă la furnizarea soluţiilor) are o importanţă destul de mare, chiar esenţială, în cazurile cînd utilitatea unei soluţii este limitată în timp. În aceste condiţii, deşi un algoritm este corect atît din punct de vedere formal, cît şi al metodei el poate fi ineficient, în sensul că, de exemplu, furnizează soluţia mai tîrziu decît momentul în care avem nevoie de ea. De asemenea, un algoritm poate fi ineficient şi sub alte aspecte (precizie, volum de date necesare, resurse utilizate, etc.). În consecinţă eficienţa este şi ea o condiţie importantă ce trebuie luată în considerare la întocmirea unui algoritm.

2.2 Descrierea algoritmilor

De obicei, cum este şi cazul lucrării de faţă, scopul întocmirii unui algoritm este transpunerea sa pe calculator. Această transpunere se poate face direct, folosind un limbaj de programare sau parcurgînd una sau mai multe etape intermediare. Exprimarea directă într-un limbaj de programare pe lîngă faptul că necesită o oarecare experienţă, este şi destul de dificilă, mai ales în cazul algoritmilor complecşi. De aceea este de preferat parcurgerea uneia sau mai multor etape intermediare între întocmirea algoritmului în limbaj natural şi transpunerea sa într-un limbaj de programare.

Aceste etape intermediare dau posibilitatea descrierii algoritmului fără a ne concentra în mod deosebit atenţia asupra rigorilor impuse de un limbaj de programare, ci asupra corectitudinii întocmirii acestuia.

Există mai multe metode de descriere a algoritmilor: **schemele logice, pseudocodul, tabelele de decizii, diagramele, limbajele specializate de descriere**, etc. Din acestea ne oprim numai asupra primelor două.

● Schemele logice

Schemele logice sînt *forme de reprezentare grafică a algoritmilor folosind simboluri a căror formă indică tipul acţiunii.* Aceste simboluri poartă numele de *blocuri*. În interiorul blocurilor, de la caz la caz, pot figura una sau mai multe operaţii care descriu efectiv acţiunea ce se execută.

Deşi există o multitudine de simboluri utilizate în descrierea algoritmilor, blocurile fundamentale necesare întocmirii unei scheme logice sînt prezentate în figura 2.3.

Blocul terminal pune în evidenţă începutul şi sfîrşitul unei scheme logice; spre deosebire de celelalte, blocul terminal START, respectiv STOP, pot să apară numai o singură dată într-o schemă logică. Deci, o schemă logică nu poate avea decît un singur început, respectiv un singur sfîrşit.

Blocul de intrare / ieşire pune în evidenţă informaţiile de intrare, respectiv ieşire, precum şi momentul şi locul în care acestea se execută; prin intermediul acestor blocuri are loc schimbul de informaţii între algoritmul propriu-zis şi exterior.

Blocul de calcul pune în evidenţă operaţiile aritmetice, precum şi transferul de valori între diferitele variabile utilizate în algoritm. Atît transferul de valori între diferitele variabile, cît şi efectuarea operaţiilor aritmetice şi logice, presupun existenţa unor variabile *sursă* (cu care se efectează operaţia) şi a unei variabile *destinaţie* (care primeşte valoarea expresiei calculate). Operaţia prin care o variabilă primeşte rezultatul evaluării unei expresii se numeşte 'atribuire' şi se poate evidenţia utilizînd diferite simboluri. În figura 2.4 sînt prezentate trei moduri echivalente prin care se simbolizează că variabilei `z` i se atribuie valoarea produsului variabilelor `x` şi `y`, iar variabilei `k`, valoarea `k+1` (valoarea sa este crescută cu o unitate).

Observaţie. *Semnul `'='` utilizat în figura 2.4c (semnificînd o atribuire) nu trebuie confundat cu cel utilizat în matematică (semnificînd egalitatea). Astfel dacă din punct de vedere al operaţiei de atribuire `k=k+1` este o propoziţie adevarată, în matematică, ea este falsă.*

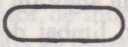
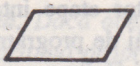
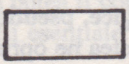
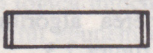
SIMBOL	SEMNIFICATIE	EXEMPLE	OBSERVAȚII
	bloc terminal	START STOP	semnifică începutul schemei logice (are, numai ieșire) semnifică terminarea schemei logice (are numai intrare)
	bloc de intrare / ieșire	CITEȘTE x,y SCRIE z	o intrare și o ieșire
	bloc de calcul	z=x+y k=j+4 x=3y	o intrare și o ieșire
	bloc de decizie	NU z>0 DA	o intrare și mai multe ieșiri
	bloc de procedură	SUMA (x,y,z)	o intrare și o ieșire
	bloc conector		mai multe intrări și o singură ieșire
	săgeata	Evidențiază sensul de parcurgere (succesiune) a elementelor schemei logice.	

Fig. 2.3

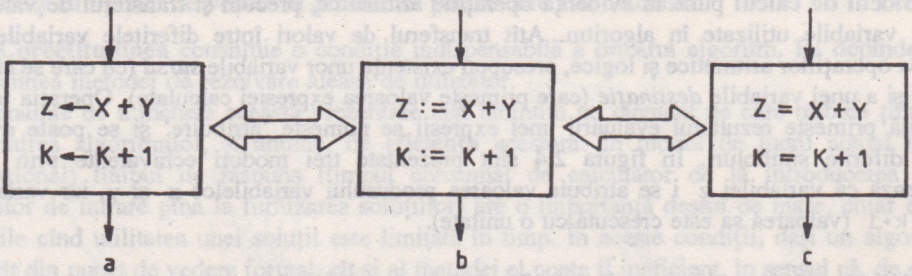


Fig. 2.4

Deoarece în limbajul BASIC, în operațiile de atribuire se utilizează semnul '=', și pentru a-l deosebi totuși de semnul '=' utilizat în matematică, în cele ce urmează vom folosi la simbolizarea

atribuirii semnul compus ':='.

Blocul de decizie pune în evidenţă punctele de ramificare ale algoritmului funcţie de anumite condiţii.

Blocul de procedură (funcţie) permite scrierea prescurtată a unei secvenţe care execută o anumită funcţiune; în interiorul blocului de procedură se scriu, pe lîngă numele simbolic al acesteia, variabilele de intrare şi de ieşire fără a face însă vreo referire la operaţiile care au loc în interiorul său.

Blocul conector pune în evidenţă nodurile (punctele de intersecţie) unui algoritm; spre deosebire de blocurile de decizie care pun în evidenţă căile alternative de urmat la un moment dat funcţie de anumite condiţii, nodurile sînt puncte în care se ajunge parcurgînd una sau alta din căile deschise de un bloc de decizie; deoarece, aşa cum ştim, un algoritm are un singur punct de început şi unul de sfîrşit (marcate prin blocurile terminale respective) existenţa în cadrul acestuia a unui bloc de decizie implică automat existenţa a cel puţin unui bloc conector.

Modul de parcurgere al algoritmului (de trecere de la un bloc la altul) este indicat prin **săgeţi** avînd o singură direcţie (un singur vîrf). Legătura între blocuri nu poate fi deci decît univocă (condiţie impusă de însăşi noţiunea de algoritm).

Exemplul 2.3

O problemă foarte simplă: produsul a două numere reale. Schema logică a soluţionării problemei este prezentată în figura 2.5.

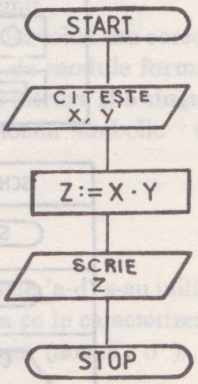


Fig. 2.5

Pe lîngă blocurile fundamentale descrise pînă acum, în figura 2.6 sînt prezentate şi alte simboluri, utilizate în special pentru a desemna **suportul extern** de pe care, respectiv pe care, are loc **citirea**, respectiv **scrierea** informaţiilor.

● Limbajul algoritmic 'pseudocod'

Pseudocodul poate fi definit ca un *limbaj simbolic situat între limbajul natural şi limbajul de programare, obţinut prin introducerea unor reguli specifice limbajelor de programare în limbajul natural.*

El operează cu un set de **cuvinte simbolice** cu semnificaţii stricte utilizate în **construcţii sintactice** care simulează structurile necesare descrierii algoritmilor. De regulă, în descrierea algoritmilor aceste cuvinte se scriu îngroşate sau subliniate, cu o aliniere distinctă faţă de restul textului. Aceste cuvinte mai poartă şi numele de **cuvinte cheie**.

Între blocurile utilizate în schemele logice şi cuvintele cheie pseudocod există o oarecare echivalenţă (figura 2.7).

În figura 2.8 se prezintă descrierea în pseudocod a algoritmului de calcul al produsului `z=x·y` (reprezentat şi prin schema logică din figura 2.5).

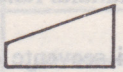
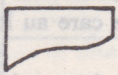
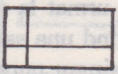
SIMBOL	SEMNIFICAȚIE	BLOC ECHIVALENT	FUNCȚIE
	tastatură	intrare (citire)	introducerea (citirea) datelor de la tastatură
	imprimantă	ieșire (scriere)	obținerea situațiilor (scrierea) la imprimantă
	ploter	ieșire (scriere)	obținerea situațiilor grafice la ploter
	monitor (display)	ieșire (scriere)	obținerea situațiilor grafic sau alfanumeric pe ecran
	bandă magnetică (casetă magnetică)	intrare/ieșire	citirea/scrierea informațiilor într-un/dintr-un fișier pe bandă magnetică
	disc magnetic fics	intrare/ieșire	citirea/scrierea informațiilor într-un/dintr-un fișier pe disc magnetic
	floppy disk (dischetă)	intrare/ieșire	

Fig. 2.6

SCHEME LOGICE	PSEUDOCOD
START	algoritm „nume" sau procedură „nume" etc.
STOP	stop
CITEȘTE a,b	citește a,b;
SCRIE r	scrie r;
r:=a+b	r:=a+b ;
PRODUS (a,b,c)	produs (a,b,c)

Fig. 2.7

```
algoritm produs x y ;
citeste  x,y ;
z:=x·y ;
scrie z ;
stop
```

Fig. 2.8

Observaţie. Caracterul ';' constituie un semn de punctuaţie foarte important în descrierea în pseudocod a unui algoritm. El este utilizat pentru separarea expresiilor, lipsa lui putînd genera erori. Caracterul '.' semnifică alături de 'stop', sfîrşitul algoritmului. Cu aceste semne de punctuaţie pentru descrierea unui algoritm în pseudocod este necesară şi suficientă o singură frază.

Reprezentarea algoritmilor în pseudocod se caracterizează printr-o mare flexibilitate, posibilitatea exprimării algoritmilor în limbaj natural şi, dacă este bine utilizat, o conversie facilă într-un limbaj de programare.

Spre deosebire de schemele logice, pseudocodul este mult mai 'bogat', el posedînd expresii care, aşa cum se va vedea în capitolul 2.3, simulează structurile fundamentale. El obligă utilizatorul la conceperea unor algoritmi structuraţi (fără salturi) ceea ce duce la o creştere substanţială a calităţii acestora şi oferă, în acelaşi timp o bună lizibilitate şi supleţe algoritmilor.

2.3 Structuri fundamentale

În cele prezentate pînă acum ne-am oprit şi asupra cîtorva exemple de algoritmi (capitolul 2.1). Prezentarea acestora a fost efectuată în scopul familiarizării utilizatorului cu modul de abordare a unei probleme şi cu noţiunea de algoritm, din care cauză descrierea acestora a fost efectuată utilizînd limbajul natural (prin enumerarea paşilor şi a operaţiilor de executat în fiecare moment).

În practica programării însă algoritmii se întocmesc folosind o metodă de reprezentare (capitolul 2.2) cu respectarea anumitor reguli şi utilizînd un set de structuri fundamentale, astfel încît în final acesta să posede proprietăţile fundamentale ale unui algoritm bine întocmit.

Pentru ca un algoritm să fie simplu, eficient, intuitiv uşor de verificat şi dezvoltat sau corectat (dacă este cazul) încă de la început el trebuie realizat ca o succesiune liniară de module formate din una sau mai multe instrucţiuni (acţiuni), cu proprietatea că au o singură intrare şi o singură ieşire. Astfel, dacă avem de reprezentat o secvenţă oarecare (pe care să o notăm simbolic 'S') într-un algoritm în care se determină valorile:

$$A = x + y$$
$$B = 2 \cdot x + y^3$$
$$C = x^2 + y^2$$

putem folosi oricare din variantele prezentate în figura 2.9, în care pentru variantele 'a-d' s-au utilizat unul sau mai multe blocuri de calcul, iar în varianta 'e', un bloc procedură. Ceea ce le caracterizează însă pe toate este existenţa unei singure intrări (notată 'i') şi a unei singure ieşiri (notată 'o').

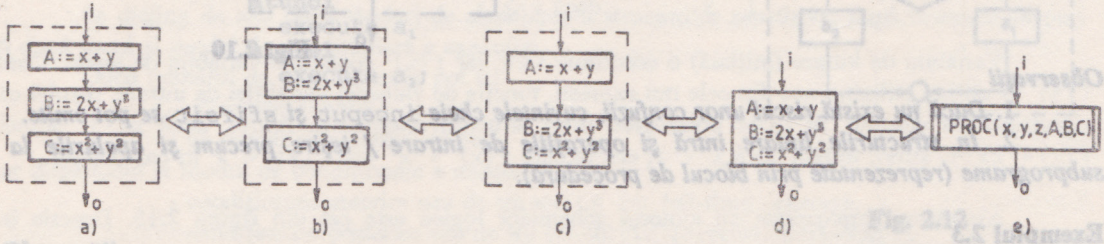


Fig. 2.9

În practică însă, în întocmirea algoritmilor întîlnim şi alte condiţii de derulare. Sînt astfel situaţii care impun, funcţie de anumite condiţii, execuţia uneia sau altei secvenţe de operaţii, precum şi altele în care una sau mai multe operaţii se execută pînă se îndeplinesc anumite condiţii, situaţii care pot determina mai multor ieşiri dintr-un modul. Cum putem ieşi din această situaţie? Vom organiza de aşa natură acţiunile în fiecare modul, astfel încît, în final să obţinem o singură intrare şi o singură ieşire.

Pentru aceasta dispunem de trei **structuri fundamentale**:

- structuri liniare;
- structuri alternative;
- structuri repetitive.

Aceste structuri constituie module avînd importanta proprietate amintită (o singură intrare şi o singură ieşire).

Noţiunea de modul este însă mai largă, în sensul că un modul poate conţine una sau mai multe structuri, deci poate fi, la rîndul său, alcătuit din mai multe submodule. În descrierea structurilor fundamentale vom considera însă că un modul conţine o singură astfel de structură.

● Structuri liniare

Structurile liniare sînt succesiuni de operaţii (acţiuni) care se execută necondiţionat în ordinea dată. O structură liniară care conţine `n` operaţii notate a_1, a_2, ..., a_n poate fi reprezentată, utilizînd schemele logice, ca în figura 2.10.

În pseudocod această structură se poate reprezenta astfel:

```
inceput
        executa a1;
        executa a2;
        .
        .
        .
        executa an
sfirsit.
```

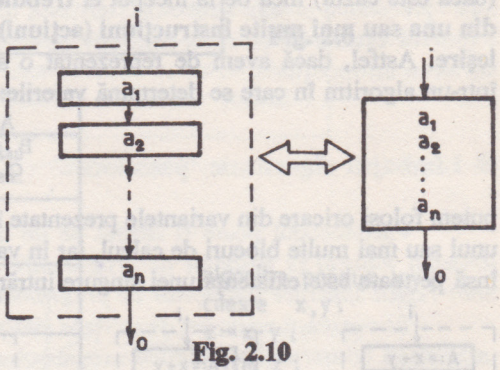


Fig. 2.10

Observaţii

1. *Dacă nu există riscul unor confuzii, cuvintele cheie* `inceput` *şi* `sfirsit` *se pot omite.*
2. *În structurile liniare intră şi operaţiile de intrare / ieşire precum şi apelurile la subprograme (reprezentate prin blocul de procedură).*

Exemplul 2.3

Să reprezentăm utilizînd schemele logice şi în pseudocod algoritmul pentru calculul valorilor variabilelor `A`, `B`, `C` prezentate în figura 2.9. Algoritmul complet pentru determinarea valorilor acestor

variabile este prezentat în figura 2.11. Variabilele de intrare sînt x, y, z R (mulţimea valorilor admisibile este deci mulţimea numerelor reale), iar variabilele de ieşire sînt A, B, C.

După cum se poate observa întregul algoritm utilizează numai structuri de tip liniar.

Descris în pseudocod, algoritmul este:

```
algoritm calcul_ABC;
    citeste x,y,z ;
    A := x + y ;
    B := 2 x + y^3 ;
    C := x^2 + y^2 ;
    scrie A,B,C
stop.
```

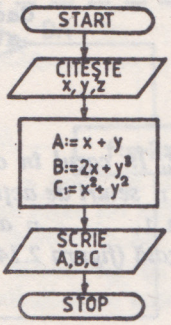


Fig. 2.11

● Structuri alternative

Cunoscute şi sub numele de structuri IF-THEN-ELSE (DACA-ATUNCI-ALTFEL), şi avînd forma generală prezentată în figura 2.12, sînt alcătuite dintr-un bloc de decizie (c) şi două blocuri a_1 şi a_2 care se execută alternativ (sau a_1 sau a_2), după cum c este adevarată sau falsă (DA / NU), şi un bloc conector.

Într-o structură alternativă derularea acţiunilor decurge astfel:

- se intră în modul i;
- se evaluează expresia c şi:
  - dacă este adevarată se execută a_1;
  - dacă nu, se execută a_2;
- se iese din modul o.

După cum se poate observa, deşi în interiorul modulului există două ramuri disjuncte el are totuşi o singură intrare şi o singură ieşire. Lucrul acesta este posibil deoarece structura alternativă începe printr-un bloc de decizie (care are o singură intrare) şi se încheie printr-un bloc conector care asigură conexiunea ramurilor alternative.

În pseudocod o structură alternativă se reprezintă sub forma

```
daca c
  atunci
    executa a1
  altfel
    executa a2;
```

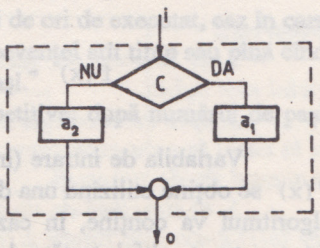


Fig. 2.12

Observaţii

1. *Într-o structură alternativă nu este obligatorie existenţa ambelor blocuri a_1 şi a_2. Dacă, de exemplu a_2 nu conţine operaţii (este vidă) atunci se obţine structură alternativă cu o ramură vidă*

(figura 2.13), care în pseudocod este:

```
daca c
 atunci
   executa a1;
```

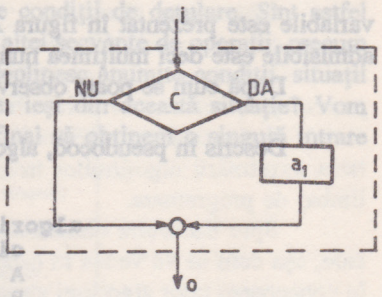


Fig. 2.13

2. În cazul în care selecţia se execută dintr-o mulţime finită de n seturi de acţiuni, după un comutator k, ce ia valori în mulţimea 1, ..., n atunci se obţine o structură alternativă generalizată (figura 2.14), care în pseudocod se poate reprezenta astfel:

```
selecteaza k din
   1: executa a1;
   2: executa a2;
      .
      .
      .
   n: executa an
sfirsit.
```

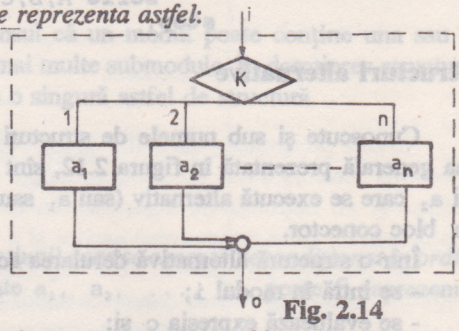


Fig. 2.14

La acest tip de structură alternativă cuvîntul cheie `sfirsit` este obligatoriu.

3. Blocurile a_1 respectiv a_2 pot conţine la rîndul lor structuri alternative, deci o structură alternativă poate conţine la rîndul său structuri alternative.

Exemplul 2.4

Să se reprezinte cu ajutorul schemelor logice şi în pseudocod algoritmul de calcul al valorilor funcţiei:

$$f : R \to R$$

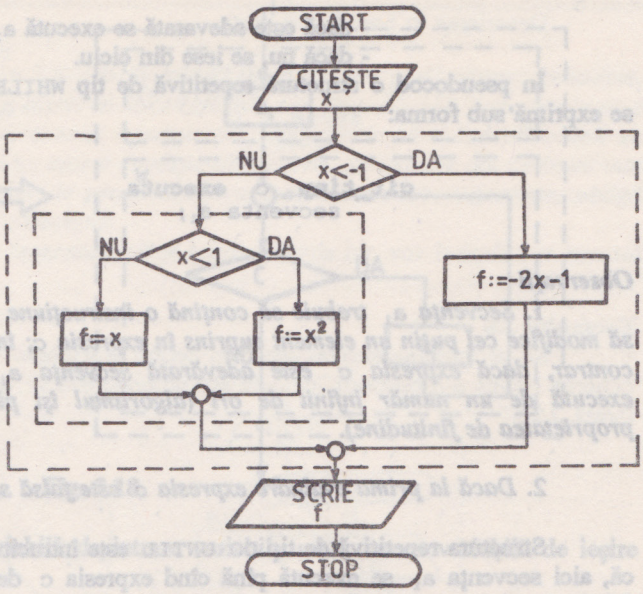
$$f(x) = \begin{cases} -2x - 1 & \text{dacă } x < -1 \\ x^2 & \text{dacă } -1 \le x < 1 \\ x & \text{dacă } x \ge 1 \end{cases}$$

Variabila de intrare (iniţială) o constituie $x \in R$, iar $f(x)$ constituie variabila de ieşire. Cum $f(x)$ se obţine utilizînd una din cele trei expresii, funcţie de valoarea variabilei de intrare, rezultă că algoritmul va conţine, în cazul considerat, două structuri alternative. Astfel, dacă $x<-1$ atunci $f(x)=-2x-1$, altfel, testăm dacă $x < 1$, caz în care $f(x) = x^2$ sau $f(x) = x$ în caz contrar.

Algoritmul reprezentat cu ajutorul schemelor logice este cel din figura 2.15. Descris în pseudocod algoritmul este:

```
algoritm functie
 citeste x ;
```

```
daca (x < -1)
  atunci
    f := -2x-1
  altfel
    daca (x<1)
      atunci
        f := x²
      altfel
        f := x
      sfirsit ;
  sfirsit ;
  scrie f ;
stop.
```



Observăm că a doua structură alternativă este inclusă în prima, dar că, fiecare, privită din exteriorul său, îşi păstrează proprietatea fundamentală de a avea o singură intrare şi o singură ieşire.

Fig. 2.15

Remarcă. *Utilizarea de două ori a cuvîntului cheie* `sfirsit` *pentru a desemna încheierea celor două structuri alternative nu este absolut necesară; au fost însă folosite pentru a mări lizibilitatea algoritmului.*

● Structuri repetitive

Structurile repetitive permit efectuarea uneia sau a mai multor operaţii (acţiuni) de un număr de ori sau pînă cînd se îndeplineşte o anumită condiţie. Structurile repetitive mai poartă şi denumirea de structuri ciclice sau cicluri.

Aceste structuri presupun fie cunoaştera şi precizarea numărului de ori de executat, caz în care avem de-a face cu **un ciclu cu număr cunoscut de paşi**, fie execuţia secvenţei atît timp sau pînă cînd o anumită condiţie este îndeplinită, **ciclu cu număr necunoscut de paşi**.

Se disting de aici două moduri de clasificare a structurilor repetitive: după numărul de paşi şi după locul şi modul de condiţionare a execuţiei.

După numărul de paşi, ciclurile pot fi:

- cu număr cunoscut de paşi;
- cu număr necunoscut de paşi,

iar după locul şi modul de condiţionare a execuţiei sînt:

- condiţionate anterior sau de tip `WHILE-DO` (cît-timp...execută...);
- condiţionate posterior sau de tip `DO-UNTIL` (execută...pînă-cînd...).

Structura de tip `WHILE-DO` este prezentată în figura 2.16 şi se execută astfel:

- se evaluează expresia c şi :

- dacă este adevarată se execută a_1 şi se reia;
- dacă nu, se iese din ciclu.

În pseudocod o structură repetitivă de tip WHILE-DO se exprimă sub forma:

```
cit_timp  c  executa
      secventa a1;
```

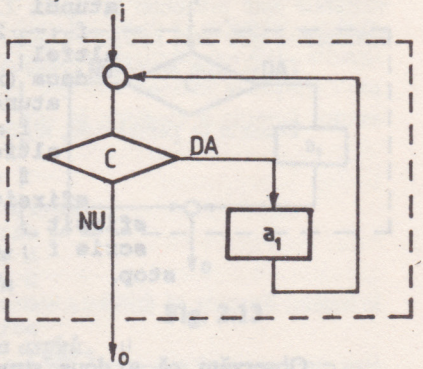


Fig. 2.16

Observaţii

1. *Secvenţa a_1 trebuie să conţină o instrucţiune care să modifice cel puţin un element cuprins în expresia c; în caz contrar, dacă expresia c este adevărată secvenţa a_1 se execută de un număr infinit de ori (algoritmul îşi pierde proprietatea de finitudine).*

2. *Dacă la prima evaluare expresia c este falsă secvenţa a_1 nu se execută niciodată.*

Structura repetitivă de tip DO-UNTIL este întrucîtva inversă structurii WHILE-DO, în sensul că, aici secvenţa a_1 se execută pînă cînd expresia c devine adevărată (deci atît timp cît c este falsă), locul în care se face această evaluare fiind de data aceasta la sfîrşit. Structura repetitivă DO-UNTIL este prezentată în figura 2.17 şi se execută astfel:

- se execută secvenţa a_1;
- se evaluează expresia c şi:
  - dacă este falsă se reia secvenţa a_1;
  - dacă este adevărată se iese din ciclu.

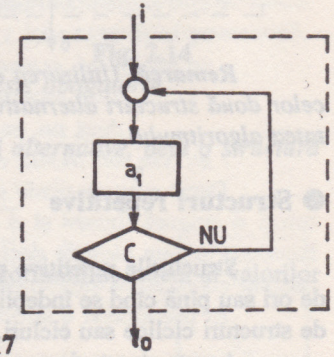


Fig.2.17

În pseudocod această structură se exprimă:

```
executa
      secventa a1
pina_cind c.
```

Observaţii

1. *Şi în acest caz secvenţa a_1 trebuie să determine modificarea condiţiei c; în caz contrar ciclul este infinit.*

2. *Spre deosebire de structura WHILE-DO aici secvenţa a_1 se execută cel puţin o dată (chiar şi în cazul în care expresia c este adevărată); din această cauză se recomandă utilizarea cu atenţie a acestui tip de structură.*

3. *O structură de tip DO-UNTIL poate fi simulată utilizînd o structură liniară (1) şi una de tip WHILE-DO (w), figura 2.18.*

Exemplul 2.5

Să se descrie utilizînd schemele logice şi în pseudocod, un algoritm pentru determinarea sumei primelor n numere naturale.

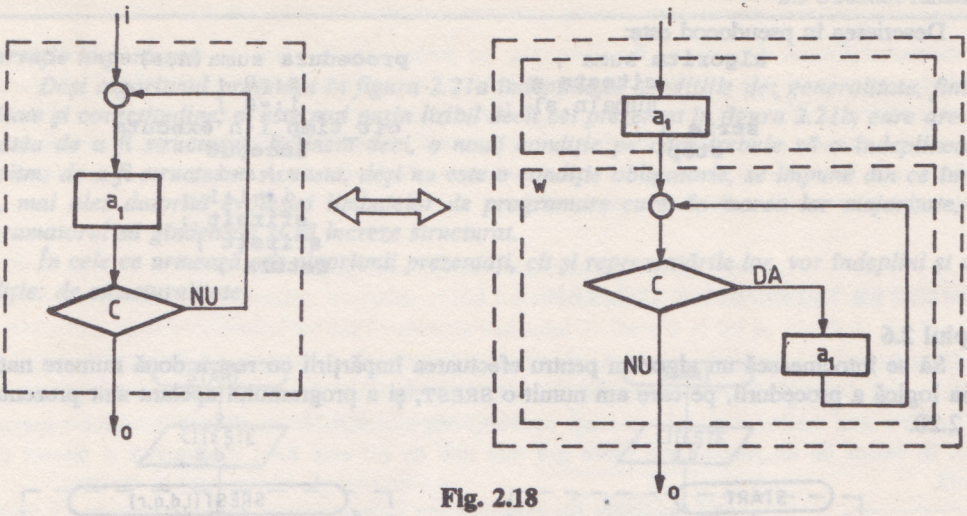


Fig. 2.18

Pentru aceasta vom utiliza ca variabilă de intrare variabila `n`$\in$`N`, iar ca variabilă de ieşire variabila $s=\sum_{i=1}^{n-1} i$ (deoarece primul număr natural este 0, atunci primele `n` numere naturale vor fi cuprinse în intervalul `[0,n-1]`).

Numărul `n` fiind cunoscut se va utiliza pentru determinarea sumei o structură cu număr cunoscut de paşi. Suma `s` se determină în procedura `SUMA`.

Schema logică a algoritmului este prezentată în figura 2.19 blocurile terminale `SUMA(n,s)` şi respectiv `RETURN` reprezintă intrarea respectiv revenirea din procedură.

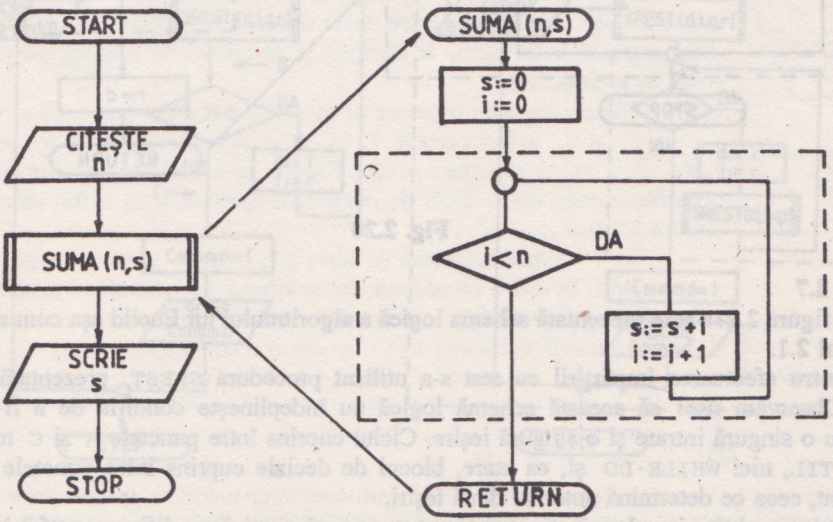


Fig. 2.19

Descrierea în pseudocod este:

```
algoritm suma ;
        citeste n ;
        suma(n,s) ;
scrie s ;
  stop.
```

```
procedura suma(n,s)
        s:=0 ;
        i:=0 ;
cit_timp i<n executa
       inceput
        s:=s+i ;
        i:=i+1 ;
       sfirsit ;
      sfirsit ;
      return ;
```

Exemplul 2.6

Să se întocmească un algoritm pentru efectuarea împărţirii cu rest a două numere naturale. Schema logică a procedurii, pe care am numit-o SREST, şi a programului apelant sînt prezentate în figura 2.20.

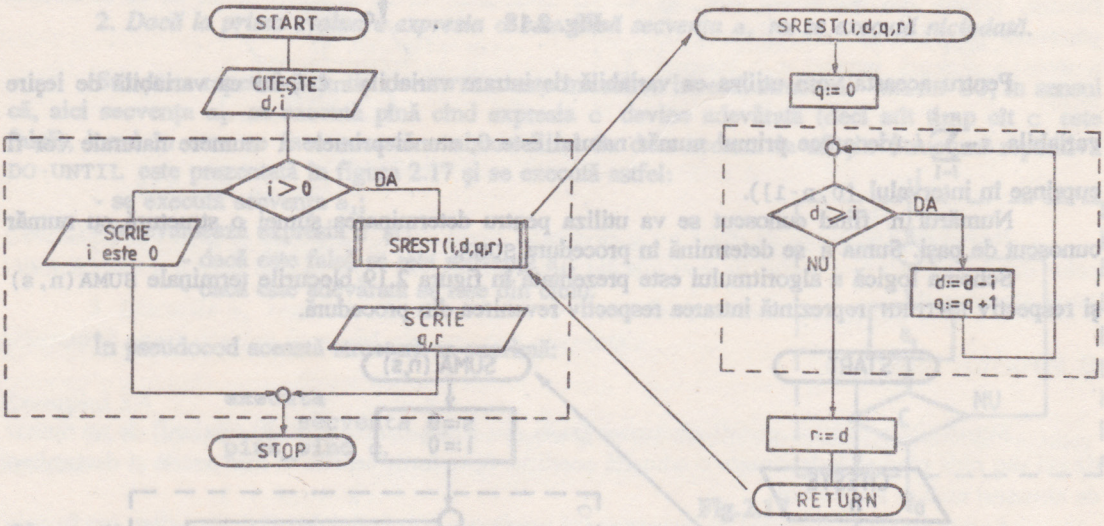


Fig. 2.20

Exemplul 2.7

În figura 2.21a este prezentată schema logică a algoritmului lui Euclid aşa cum a fost întocmit în exemplul 2.1.

Pentru efectuarea împărţirii cu rest s-a utilizat procedura SREST prezentată în exemplul anterior. Observăm uşor că această schemă logică nu îndeplineşte condiţia de a fi alcătuită din structuri cu o singură intrare şi o singură ieşire. Ciclul cuprins între punctele A şi C nu este nici de tip DO-UNTIL, nici WHILE-DO şi, ca atare, blocul de decizie cuprins între punctele B şi C este independent, ceea ce determină automat două ieşiri.

Pentru a obţine o schemă structurată este necesară o mică modificare, astfel încît se obţine schema logică din figura 2.21b.

Observație importantă

Deși algoritmul prezentat în figura 2.21a îndeplinește condițiile de: generalitate, finitudine, unicitate și corectitudine, el este mai puțin lizibil decît cel prezentat în figura 2.21b, care are în plus calitatea de a fi structurat. Reținem deci, o nouă condiție pe care trebuie să o îndeplinească un algoritm: de a fi structurat. Aceasta, deși nu este o condiție obligatorie, se impune din ce în ce mai mult, mai ales datorită evoluției limbajelor de programare care, în marea lor majoritate, obligă programatorul să gîndească și să lucreze structurat.

În cele ce urmează atît algoritmii prezentați, cît și reprezentările lor, vor îndeplini și această condiție: de structuralitate.

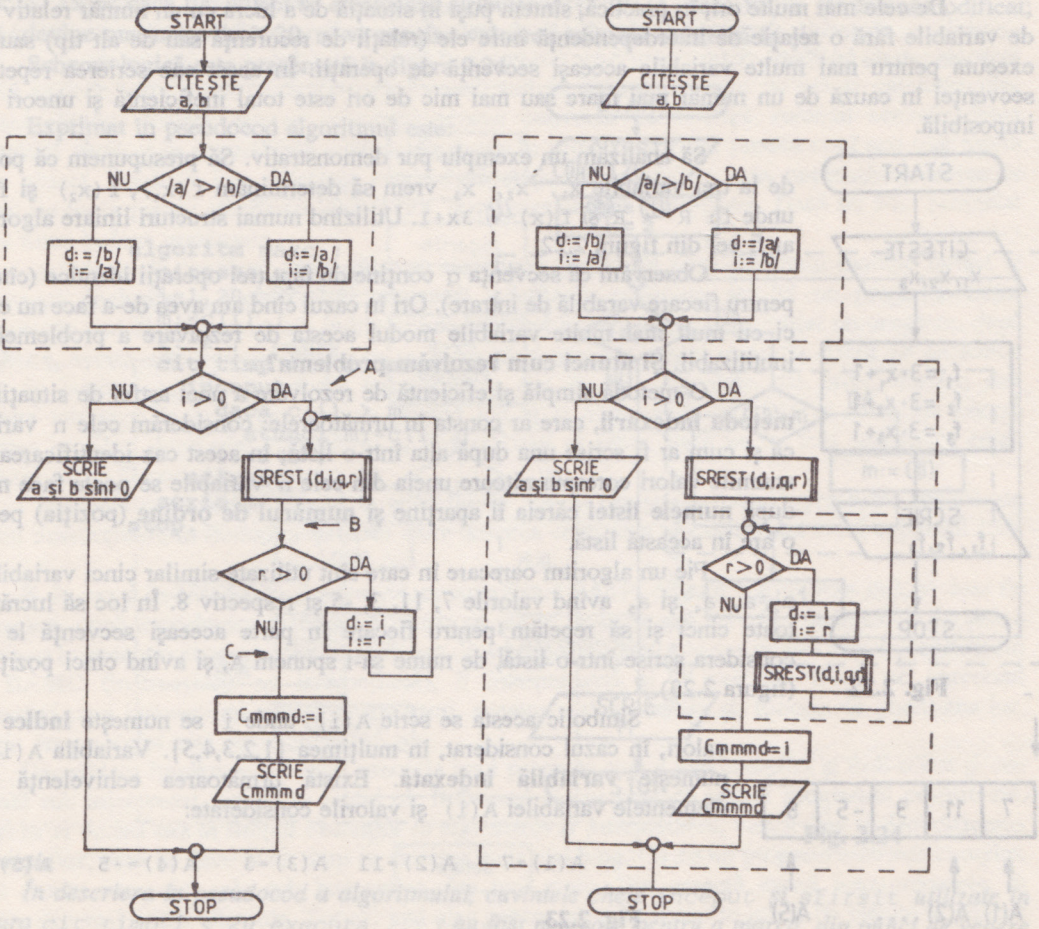


Fig. 2.21

Remarcă finală. Structurile fundamentale nu trebuie confundate cu blocurile fundamentale utilizate în reprezentarea algoritmilor cu ajutorul schemelor logice. Dacă blocurile de calcul, blocurile de intrare / ieşire şi blocurile procedură pot fi considerate şi ca structuri de tip liniar, blocul de decizie, avînd o intrare şi cel puţin două ieşiri nu poate fi asimilat cu nici una din structuri. În mod asemănător stau lucrurile şi în cazul blocului conector. Blocurile de decizie şi conector intră însă în compunerea structurilor alternative şi repetitive.

2.4 Utilizarea variabilelor indexate

De cele mai multe ori, în practică, sîntem puşi în situaţia de a lucra cu un număr relativ mare de variabile fără o relaţie de interdependenţă între ele (relaţii de recurenţă sau de alt tip) sau de a executa pentru mai multe variabile aceeaşi secvenţă de operaţii. În acest caz scrierea repetată a secvenţei în cauză de un număr mai mare sau mai mic de ori este total ineficientă şi uneori chiar imposibilă.

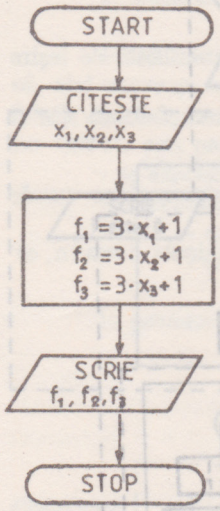


Fig. 2.22

Să analizăm un exemplu pur demonstrativ. Să presupunem că pornind de la trei variabile x_1, x_2, x_3 vrem să determinăm $f(x_1)$, $f(x_2)$ şi $f(x_3)$ unde $f: R \to R$ şi $f(x) = 3x+1$. Utilizînd numai structuri liniare algoritmul ar fi cel din figura 2.22.

Observăm că secvenţa `q` conţine de fapt trei operaţii identice (cîte una pentru fiecare varabilă de intrare). Ori în cazul cînd am avea de-a face nu cu trei ci cu mult mai multe variabile modul acesta de rezolvare a problemei este inutilizabil. **Şi atunci cum rezolvăm problema?**

O metodă simplă şi eficientă de rezolvare a unei astfel de situaţii este **metoda indexării,** care ar consta în următoarele: considerăm cele `n` variabile ca şi cum ar fi scrise una după alta într-o **listă;** în acest caz identificarea unei anumite valori corespunzătoare uneia din cele `n` variabile se poate face numai după **numele listei** căreia îi aparţine şi **numărul de ordine** (poziţia) pe care o are în această listă.

Fie un algoritm oarecare în care sînt utilizate similar cinci variabile a_1, a_2, a_3, a_4 şi a_5 avînd valorile 7, 11, 3, -5 şi respectiv 8. În loc să lucrăm cu toate cinci şi să repetăm pentru fiecare în parte aceeaşi secvenţă le vom considera scrise într-o listă, de nume să-i spunem `A`, şi avînd cinci poziţii (figura 2.23).

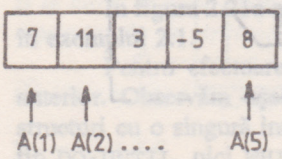


Fig. 2.23

Simbolic acesta se scrie `A(i)` unde `i` se numeşte **indice** şi ia valori, în cazul considerat, în mulţimea {1,2,3,4,5}. Variabila `A(i)` se numeşte **variabilă indexată**. Există următoarea echivelenţă între elementele variabilei `A(i)` şi valorile considerate:

`A(1)=7  A(2)=11  A(3)=3  A(4)=-5  A(5)=8.`

O variabilă indexată poate avea unul sau mai mulţi indici. Variabilele indexate cu un singur indice se mai numesc **tablouri unidimensionale** sau **vectori**, iar cele cu doi indici, de exemplu `B(i,j)`, **tablouri bidimensionale** sau **matrici**.

Exemplul 2.8

Să se întocmească un algoritm pentru determinarea elementului maxim dintr-un şir de 20 de valori date.

Pentru aceasta utilizăm variabila indexată `C(i)` unde:

$$i \in \{ i \mid 1 \leq i \leq 20 \text{ şi } i \in N \}.$$

Variabila `C(i)` constituie variabila de intrare. Variabila de ieşire este `m = maxC(i)`. Pentru determinarea elementului maxim, atribuim iniţial lui `m` valoarea primului element din listă (vectorul) `C ( m=C(1))`, apoi modificînd pe rînd valoarea indicelui `i` între 2 şi 20 se verifică, de fiecare dată, dacă `C(i) > m`; dacă DA atunci se efectuează atribuire `m := C(i)`, dacă NU, `m` rămîne nemodificat; cînd `i` devine mai mare decît 20, `m` va conţine valoarea maximă existentă în şir.

Schema logică este prezentată în figura 2.24.

Exprimat în pseudocod algoritmul este:

```
algoritm maxc ;
  citeste c(i) ;
  i=1,20
  m:=c(i) ;
  i:=2 ;
  cit_timp i < 20 executa
    inceput
      daca c(i) > m
        atunci m:=c(i) ;
      i:=i+1 ;
    sfirsit ;
  scrie m ;
stop.
```

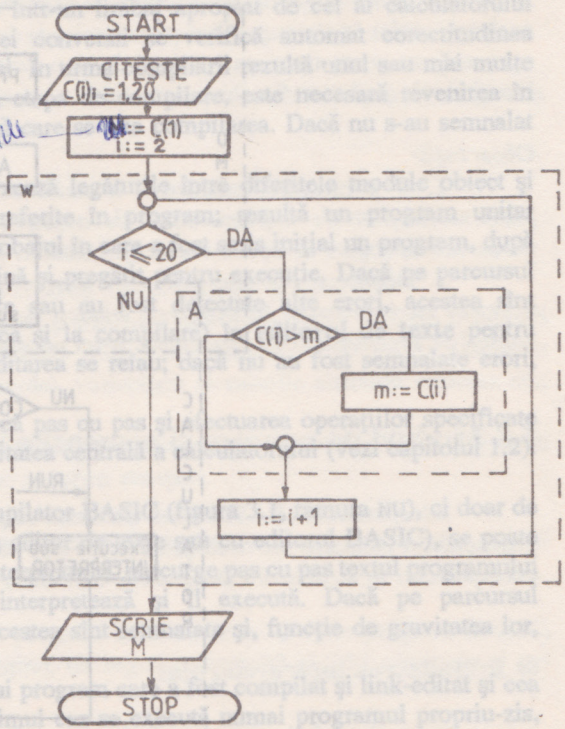


Fig. 2.24

Observaţie

În descriera în pseudocod a algoritmului, cuvintele cheie `inceput` şi `sfirsit` utilizate în structura `cit_timp i < 20 executa ...`, au fost necesare pentru a marca, din punct de vedere formal, faptul ca ambele expresii cuprinse între ele (reprezentînd o structură alternativă şi una liniară) se repetă atît timp cît condiţia `i < 20` este îndeplinită. Dacă acestea ar fi lipsit, ar fi rezultat că se repetă numai prima expresie (respectiv structura alternativă), ceea ce este eronat.

3. ELEMENTE ALE LIMBAJULUI BASIC

3.1 Cîteva precizări

În general, scopul final al elaborării unui algoritm este transpunerea şi execuţia automată a acestuia cu ajutorul calculatorului. În esenţă, etapele de parcurs pentru îndeplinirea acestui deziderat sînt prezentate în figura 3.1.

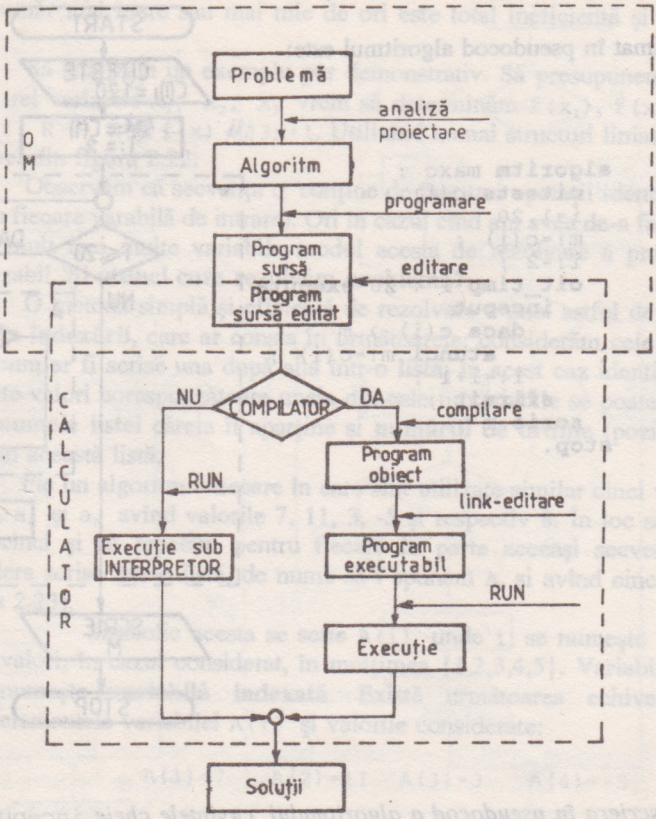


Fig. 3.1

Am văzut, în capitolele anterioare, care sînt principalele cerinţe şi cum putem întocmi algoritmul soluţionării unei probleme.

După cum rezultă şi din figura 3.1, pentru a transpune algoritmul pe calculator, pentru a putea

fi 'înţeles' şi executat de acesta, ar mai fi de parcurs cel puţin încă o etapă, şi anume, **descrierea sa într-un limbaj de programare**. Am ales limbajul BASIC.

Înaine de a trece însă la descrierea efectivă a elementelor de bază ale limbajului BASIC sînt necesare cîteva precizări referitoare la fazele prin care trece un program de la introducere pînă la execuţia sa efectivă.

O primă fază constă în **editarea** (introducerea de la tastatură - scrierea) programului în format sursă (aşa cum a fost scris) şi reţinerea lui pe un suport extern de informaţii (bandă sau disc magnetic). Lucrul acesta se execută cu ajutorul unui **editor de texte**, program sistem de regulă, care permite introducerea, încărcarea şi salvarea (reţinerea) unui text (respectiv textul programului).

După editarea şi eventual corectarea textului programului, funcţie de modul de lucru (de varianta de BASIC de care dispunem, compilator sau interpretor) se poate proceda în două moduri:

A. Cînd se dispune de compilator se parcurg fazele (figura 3.1, ramura DA):

- **compilarea** programului, care constă în conversia unui program din limbajul de programare de nivel înalt în care a fost scris (program sursă) într-un limbaj apropiat de cel al calculatorului (program obiect); înainte sau pe parcursul acestei conversii se verifică automat corectitudinea programului sursă şi se semnalează eventualele erori. În urma compilării rezultă unul sau mai multe module obiect. Dacă au fost detectate erori, după etapa de compilare, este necesară revenirea în **editorul de texte** pentru efectuarea corecţiilor, după care se reia compilarea. Dacă nu s-au semnalat erori se trece la faza următoare.
- **editarea de legături** (link-editarea) realizează legăturile între diferitele module obiect şi soluţionează apelurile la funcţiile de bibliotecă referite în program; rezultă un program unitar reprezentat în cod direct executabil; indiferent de limbajul în care a fost scris iniţial un program, după link-editare, acesta este tradus complet în cod maşină şi pregătit pentru execuţie. Dacă pe parcursul editării de legături au rămas referiri nesoluţionate sau au fost detectate alte erori, acestea sînt semnalate automat; în acest caz trebuie revenit (ca şi la compilare) în **editorul de texte** pentru corectarea erorilor, după care compilarea şi link-editarea se reiau; dacă nu au fost semnalate erori, programul poate fi lansat în execuţie;
- **execuţia** programului constă în parcurgerea pas cu pas şi efectuarea operaţiilor specificate de codul instrucţiunii, în limbaj maşină, de către unitatea centrală a calculatorului (vezi capitolul 1.2).

B. În cazul în care nu se dispune de un compilator BASIC (figura 3.1, ramura NU), ci doar de **interpretor**, după introducera textului sursă (cu un editor de texte sau cu editorul BASIC), se poate trece direct la execuţia programului. În acest caz, 'interpretorul' parcurge pas cu pas textul programului sursă BASIC (instrucţiune cu instrucţiune), îl interpretează şi îl execută. Dacă pe parcursul interpretării unei instrucţiuni sînt detectate erori, acestea sînt semnalate şi, funcţie de gravitatea lor, execuţia programului este abandonată sau nu.

Deosebirea fundamentală între execuţia unui program care a fost compilat şi link-editat şi cea efectuată sub interpretor, constă în faptul că, în primul caz se execută numai programul propriu-zis, pe cînd în al doilea caz în timpul execuţiei se efectuează şi interpretarea, verificarea şi decodificarea instrucţiunilor, ceea ce duce la un timp de răspuns mult mai lung. Cu cît un program este mai complex cu atît durata execuţiei sale sub interpretor este mai mare. Din această cauză interpretoarele, la execuţii, sînt mai puţin eficiente. Totuşi, mai ales pentru începători sau în etapa de testare şi punere la punct a unor module de program sau programe, cînd durata execuţiei nu interesează în mod deosebit, interpretorul asigură o mai bună flexibilitate şi intervale de timp mai scurte între efectuarea unei corecţii în programul sursă şi verificarea eficienţei acesteia în execuţie.

În urma execuţiei programului se obţin soluţiile problemei.

Continua evoluţie a limbajelor de programare, mai ales după apariţia şi dezvoltarea de amploare a calculatoarelor personale, a dus la apariţia aşa numitelor 'medii de programare', mai întîi pentru limbajul PASCAL şi apoi treptat şi pentru alte limbaje, inclusiv BASIC.

Mediile de programare sînt sisteme integrate de programe care asigură editarea, compilarea, link-editarea, execuţia şi depanarea programelor. Ele oferă marele avantaj că, printr-un set de comenzi proprii, asigură toate funcţiile necesare punerii la punct a programelor, ceea ce duce la o programare deosebit de eficientă. Indiferent de varianta de BASIC şi de modul de lucru (sub interpretor, cu compilator sau în mediu de programare), în esenţă, nucleul vocabularului (vocabularul fundamental), semantica (setul de reguli care determină înţelesul cuvintelor - al cuvintelor cheie - şi semnificaţia propoziţiilor) şi regulile sintactice (de combinare a cuvintelor în propoziţii, şi relaţiile dintre acestea la nivelul frazei - aşa cum s-a arătat în capitolul 2, un program poate fi asimilat cu o frază) sînt aceleaşi, astfel încît, cunoscînd elementele de bază, conceptele şi structurile fundamentale cu care operează, se poate trece cu uşurinţă de la o versiune de BASIC la alta, atît din punct de vedere al însuşirii, cît şi al transferului programelor.

Observaţii:

1. *O prezentare exhaustivă a multitudinii variantelor de BASIC existente în prezent, a particularităţilor de implementare ale acestora pe diferite tipuri de calculatoare, precum şi a modurilor specifice de operare este efectuată în* [4].

2. *Deoarece obiectul prezentei lucrări nu este prezentarea limbajului BASIC ci utilizarea acestuia ca instrument în însuşirea tehnicii de programare a calculatoarelor în general, în acest capitol se prezintă numai elementele strict necesare atingerii acestui obiectiv.*

3.2 Moduri de operare şi comenzile interpretorului

Interpretoarele BASIC fiind mult mai răspîndite decît compilatoarele vom începe prezentarea principalelor elemente ale limbajului BASIC cu cîteva comenzi, minim necesare lucrului cu acesta, cu exemplificări în BASIC-80 (MBASIC, GBASIC) <B> şi GW-BASIC <G>.

● Iniţializarea interpretorului

Se realizează de obicei tastînd, ca o comandă a sistemului de operare, numele acestuia. De exemplu, apelul interpretorului MBASIC sub CP/M se efectuează cu comanda:

```
MBASIC [fis] [/F:maxf] [/M:maxm] [S:maxr]
```

unde

`fis` este numele fişierului avînd implicit tipul (extensia) `BAS` care, după încărcare, este lansat în execuţie;

`maxf` reprezintă numărul maxim de fişiere de date simultan deschise;
`maxm` este limită maximă a memoriei pe care o poate folosi interpretorul;
`maxr` reprezintă lungimea maximă a înregistrării din fişierele de date.

Interpretorul GW-BASIC, sub MS-DOS se apelează cu:

```
GWBASIC [fis][,stdin][,stdout][/F:maxf][/S:maxr]
        [/M:maxm][,lgb][/D][/I]
```

unde

`stdin` şi `stdout`
desemnează eventualele redirectări ale fişierelor standard de intrare, respectiv ieşire;
`lgb` specifică dimensiunea spaţiului de lucru rezervat pentru date şi programe încărcate peste spaţiul de lucru al lui GW-BASIC;
`D` încarcă în memorie pachetele de rutine matematice pentru funcţii transcendente;
`I` determină alocarea statică a spaţiului de memorie necesar pentru operaţii la nivel de fişier.

După iniţializare se tipăreşte prompterul `Ok` semnificînd faptul că interpretorul este operaţional.

● Moduri de operare

După lansare, interpretorul poate fi utilizat în două moduri: **direct** şi **indirect**.

Modul direct permite introducerea comenzilor şi instrucţiunilor fără număr de linie şi asigură execuţia acestora imediat după tastarea caracterului `<CR>` (`RETURN`, `ENTER`). Deşi instrucţiunile nu sînt memorate (după execuţie linia introdusă se pierde), variabilele şi valorile atribuite acestora sînt reţinute în memoria de lucru a interpretorului.

Modul de operare **direct** este util pentru depanarea unor secvenţe liniare, precum şi pentru efectuarea unor operaţii rapide, care nu necesită un program complet.

***Observaţie.** În acest mod de operare pot fi utilizate numai instrucţiunile ce nu presupun salturi de orice fel către alte linii BASIC.*

Modul indirect permite introducera unui program linie cu linie. O linie program BASIC, funcţie de familie, versiune, generaţie, poate conţine, de regulă, între 80 şi 255 caractere şi are sintaxa:

```
etich instr1 [:instr2 ...] <CR>
```

unde

`etich` este o valoare numerică întreagă cuprinsă între 1 şi o valoare maximă (32000, 32767, 65529 etc.) funcţie de versiune, şi are două utilizări: arată ordinea de memorare a liniilor program şi poate fi referită în instrucţiunile de salt;
`instr` este o propoziţie care reprezintă descriera în BASIC a unei operaţii;
`:` reprezintă semnul de punctuaţie utilizat în BASIC pentru separarea instrucţiunilor în cazul în

care sînt mai multe pe aceeaşi linie.

Sfîrşitul unei linii este semnalat de caracterul <CR>.

O linie BASIC (în majoritatea versiunilor) poate conţine mai multe linii fizice, trecerea, în acest caz, la altă linie fizică efectuîndu-se cu caracterul <LF> sau combinaţia <CTRL>J.

Observaţie. *În cazul în care într-o linie BASIC este specificată o singură instrucţiune, separatorul : nu mai este necesar, rolul său fiind îndeplinit de caracterul <CR>. Observaţia este valabilă şi în cazul existenţei mai multor instrucţiuni pe aceeaşi linie, pentru instrucţiunea ce precede caracterul <CR>.*

Numerele liniilor pot fi generate automat. Pentru aceasta se utilizează comanda:

```
                <B,G>
AUTO        [nrl][,incr]
```

Generarea începe cu numărul `nrl`, continuînd cu pasul (incrementul) `incr` (implicit `nrl=incr=10`); revenirea în mod comandă se efectuează cu <CTRL>C, respectiv <CTRL><BREAK> în GW-BASIC.

Programul introdus rămîne în memorie, asupra lui putîndu-se efectua o serie de operaţii, pînă la ştergerea sau la revenirea în sistemul de operare.

● Salvări, restaurări

Salvarea programului curent într-un fişier disc `fis` se realizează cu comanda:

```
                <B,G>
SAVE        "fis"[,A][,P]
```

Daca `fis` există, atunci programul salvat va fi scris peste acesta; opţiunea `A` determină salvarea programului în format ASCII, iar `P` duce la protecţia (la listare sau editare) fişierului salvat.

Încărcarea în memorie a unui fişier disc `fis` se efectuează cu comanda:

```
                <B,G>
LOAD        "fis"[,R]
```

Prezenţa opţiunii `R` determină, după încărcare, lansarea automată în execuţie a programului.

O altă comandă care permite încărcarea în memorie a unui fişier disc este:

```
                <B,G>
MERGE       "fis"
```

Spre deosebire de `LOAD` care determină înaintea încărcării ştergerea programului curent, `MERGE` combină programul curent cu fişierul disc `fis`. Dacă cele două programe au linii comune, liniile respective ale programului `fis` le înlocuiesc pe cele corespunzătoare din programul curent.

Observaţie importantă. *Deoarece pe parcursul lucrării de faţă vom utiliza frecvent această comandă, subliniem faptul că alături de comenzile `LOAD` şi `SAVE`, `MERGE` face parte şi din setul de*

comenzi ale următoarelor interpretoare: BASIC HC-85, TIM S, SPECTRUM, BASIC-AMSTRAD, ABASIC, MBASIC 86, ZBASIC, BASIC-TI, BETA-BASIC. În cazul microcalculatoarelor care nu dispun de unitate de dischetă, comanda `MERGE` *va determina încărcarea în memoria internă a programului* `fis` *înregistrat pe caseta magnetică. Ca şi în cazul comenzilor* `LOAD` *şi* `SAVE`*, înaintea lansării comenzii* `MERGE` *pregătirea casetofonului şi eventuala poziţionare a benzii magnetice se vor efectua conform particularităţilor de utilizare a timpului de calculator de care se dispune.*

Interpretorul GW-BASIC dispune de încă două comenzi pentru salvarea (`BSAVE`), respectiv restaurarea (`BLOAD`) unei zone de memorie.

Comanda

```
                <G>
BSAVE    "fis",depl,lg
```

salvează zona de memorie de `lg` octeţi, care începe de la adresa relativă `depl` (deplasarea în octeţi) faţă de segmentul definit de un `DEF SEG` anterior (implicit segmentul de date al interpretorului GW-BASIC), în fişierul disc `fis`.

Comanda

```
                <G>
BLOAD    "fis"[,depl]
```

încarcă în memorie fişierul în cod maşină (imagine memorie) `fis` la adresa relativă `depl` faţă de adresa segmentului definit de ultimul `DEFSEG` sau, în lipsa acestuia, faţă de adresa segmentului de date al interpretorului GW-BASIC.

● Ştergeri, modificări, listări

Ştergerea programului şi a tuturor variabilelor existente în memorie se efectuează cu comanda:

```
              <B,G>
NEW
```

Ştergerea unui fişier oarecare `fis` de pe disc se realizează cu:

```
              <B,G>
KILL     "fis"
```

Observaţie. *Fişierul* `fis` *trebuie să fie închis.*

Ştergerea uneia sau mai multor linii, avînd numerele cuprinse între `nrl1` şi `nrl2` inclusiv se realizează cu:

```
                <B,G>
DELETE      [nrl1][-][nrl2]
```

Modificarea (în mod imediat) a unei linii avînd numărul `nrl` se poate efectua cu comanda:

```
              <B,G>
EDIT        [nrl]
```

Aceasta determină intrarea în modul editare. Modificarea efectivă a textului instrucţiunii respective se realizează cu un set de subcomenzi, care în cazul interpretorului MBASIC sînt:

Spaţiu	mută cursorul la dreapta;
<RUBOUT>	mută cursorul la stînga;
I<text>	inserează 'text' pe poziţia curentă a cursorului. <text> se poate încheia cu <ESC>, caz în care editarea continuă sau cu <CR>, caz în care se iese din EDIT;
X	inserare la sfîrşit de linie;
[n]D	şterge n caractere la dreapta cursorului (inclusiv poziţia curentă);
H	şterge pînă la sfîrşitul liniei;
[n]S<ch>	caută a n-a apariţie a caracterului ch şi poziţionează cursorul înaintea acestuia;
[n]K<ch>	şterge a n-a apariţie a caracterului ch;
[n]C<ch>	înlocuieşte următoarele n caractere cu ch;
<CR>	salvează modificările făcute în linie, iese din modul EDIT şi tipăreşte restul liniei;
E	acelaşi efect ca <CR>, însă fără tipărire;
Q	dă controlul interpretorului MBASIC după salvarea modificărilor efectuate cu EDIT;
L	listează restul liniei şi poziţionează cursorul la începutul acesteia;
A	reface linia iniţială şi poziţionează cursorul la începutul ei.

Schimbarea numelui unui fişier disc se realizează cu:

```
        <B,G>
NAME      "numen" AS "numev"
```

Fişierul numev, dacă există, primeşte numele numen.
Renumerotarea liniilor se efectuează cu comanda:

```
          <B,G>
RENUM      [nrl1][,nrl2][,incr]
```

unde nrl1 reprezintă noul număr de la care începe numerotarea liniilor, număr ce se atribuie celei ce avea eticheta nrl2; incr, dacă este prezent, semnifică pasul incrementării; dacă nu, acesta este 10.

Observaţie. *RENUM actualizează şi argumentele instrucţiunilor care fac trimiteri la numere de linii mai mari sau egale cu nrl2.*

Listarea programului existent în memorie se poate efectua cu comenzile:

```
        <B>
LIST    [nrl1][-][nrl2]

        <G>
LIST    [nrl1][-][nrl2][flis]
```

```
                <B,G>
      LLIST          [nrl1][-][nrl2]
```

Acestea permit listarea unei părţi (liniile de la `nrl1` la `nrl2` inclusiv) sau a întregului program existent în memorie (`LIST` fără `nrl1` şi `nrl2`) la terminal (MBASIC), în fişierul de listare (GW-BASIC, dacă `flis` este specificat) sau la imprimantă (`LLIST`).

De asemenea, în GW-BASIC, tipărirea conţinutului ecranului pe imprimanta implicită se poate comanda cu:

```
                        <G>
      LCOPY
```

● Execuţii

Comanda

```
                <B,G>
      RUN       [nrl]
```

lansează în execuţie un program aflat în memorie începînd cu linia `nrl`. Dacă `nrl` nu este precizat atunci execuţia începe cu prima linie executabilă a programului.

În forma

```
                <B,G>
      RUN       "fis"[,R]
```

are loc ştergera conţinutului memoriei, încărcarea fişierului `fis` de pe disc şi lansarea acestuia în execuţie; dacă opţiunea `R` este prezentă, sînt păstrate toate fişierele de date deschise anterior; în caz contrar, înaintea încărcării programului `fis`, acestea sînt închise.

Lansarea în execuţie dintr-un program BASIC a altui program se poate efectua cu comanda:

```
          <B>
   CHAIN    [MERGE] "fis"[,[nrl][,ALL][,DELETE list]]
```

Aceasta apelează programul `fis`, îi transmite variabilele din programul curent şi îl lansează în execuţie începînd de la linia `nrl`; `ALL` indică faptul că toate variabilele programului curent vor fi transmise programului apelant (implicit sînt transmise numai cele declarate cu `COMMON` - vezi capitolul 3.11); `MERGE` adaugă programul `fis` la programul din memorie ca segment separat, ce poate fi suprapus peste alt segment cu opţiunea `DELETE`; `DELETE` şterge liniile specificate prin `lista` (`lista` este de forma prezentată la comanda `DELETE`).

Observaţie. *Numerele liniilor din* `lista` *sînt afectate de comanda* `RENUM`.

Comanda

```
                <G>
      SHELL     "expsir"
```

încarcă şi execută un alt prorgam sau o comandă sistem; `expsir` este preluată de MS-DOS, prelucrată, după care se revine in GW-BASIC.

● Reluări, trasări

Reluarea execuţiei unui program întrerupt cu <CTRL>C, respectiv <CTRL><BREAK> sau după execuţia unei instrucţiuni STOP sau END se poate efectua cu:

```
<B,G>
CONT
```

Dacă pe parcursul execuţiei unui program a fost detectată o eroare pentru care utilizatorul dispune de o subrutină de tratare a acesteia, după execuţia subrutinei de tratare a erorii, execuţia programului poate fi reluată cu comanda

```
<B>
RESUME    [0] [NEXT] [nrl]
```

de la linia care a produs eroarea 0, de la următoarea NEXT sau de la cea cu numărul nrl.

Comenzile

```
<B,G>
TRON
```

```
<B,G>
TROFF
```

permit activarea (TRON), respectiv dezactivarea (TROFF) 'trasării' execuţiei programului BASIC prin afişarea numerelor liniilor executate puse între paranteze drepte ('[' şi ']').

● Revenirea în sistemul de operare

Se efectuează cu comanda:

```
SYSTEM
```

care închide toate fişierele deschise şi predă controlul acestuia.

Observaţie. Pe lîngă comenzile prezentate, interpretorul GW-BASIC implementat pe microcalculatoare compatibile IBM-PC dispune şi de alte comenzi, printre care cele pentru lucrul cu cataloage, ce depăşesc însă cadrul prezentării de faţă.

3.3 Propoziţii şi fraze (programe) BASIC

Un program BASIC, ca de altfel şi în cazul celor scrise în alte limbaje de programare poate fi considerat o frază, în care instrucţiunile reprezintă propoziţii. Programul, ca orice frază, trebuie să fie coerent, clar şi precis (spre deosebire de exprimarea în limbaj natural, în limbajele de programare nu se admit ambiguităţi, subînţelesuri, etc.).

Fraza BASIC este o succesiune de linii BASIC, conţinînd una sau mai multe propoziţii (instrucţiuni) încheiată prin STOP sau END.

Liniile BASIC au sintaxa şi structura prezentată în capitolul 3.2.
Instrucţiunea STOP are formatul:

```
STOP
```

Ea determină încheierea execuţiei programului şi redă controlul interpretorului, afişînd mesajul:

```
Break in 'nrl'
```

unde nrl este numărul liniei ce conţine instrucţiunea STOP.

Instrucţiunea

```
END
```

determină, pe lîngă încheierea execuţiei programului, înainte de a reda controlul interpretorului, închiderea tuturor fişierelor deschise. END poate fi plasat oriunde în program.

Observaţie. Spre deosebire de STOP, END nu determină apariţia unui mesaj de sfîrşit la consolă.

După efectuarea comenzilor STOP şi END execuţia programului poate fi reluată din punctul de întrerupere respectiv (dacă după acesta mai există alte instrucţiuni) cu comanda CONT (capitolul 3.2).

Exemplul 3.1

Presupunem că s-a tastat, după iniţializarea interpretorului, următoarea secvenţă:

```
10   PRINT "Primul program BASIC"
20   STOP
```

unde PRINT reprezintă o instrucţiune ce va fi prezentată în capitolul 3.5 şi care aici tipăreşte la display textul cuprins între ghilimele. După tastera comenzii RUN se obţine răspunsul:

```
Primul program BASIC
Break in 20
Ok
```

Înlocuind linia 20 cu

```
20   END
```

şi retastînd RUN, obţinem răspunsul:

```
Primul program BASIC
Ok
```

Dacă în loc să folosim două linii (10 şi 20) utilizăm una singură, de exemplu:

```
10   PRINT "Primul program BASIC" : STOP
```

după comanda RUN obţinem

```
Primul program BASIC
Break in 10
Ok
```

În această ultimă variantă, cele două propoziţii `'PRINT: Primul program BASIC'` şi `'STOP'` sînt separate prin semnul de punctuaţie `':'`.

Exemplul 3.2

Să extindem puţin programul şi să plasăm instrucţiunile `STOP` şi `END` în interiorul său. Presupunem că am introdus secvenţa:

```
10   PRINT "Al doilea program BASIC"
20   PRINT "Urmeaza instructiunea STOP"
30   STOP
40   PRINT "Urmeaza instructiunea END"
50   END
60   PRINT "Sfirsitul programului"
70   END
```

în continuare conversaţia cu calculatorul decurge astfel:

```
Comanda:    RUN
Răspuns:    Al doilea program BASIC
            Urmează instrucţiunea STOP
            Break in 30
            Ok

Comanda:    CONT
Răspuns:    Urmează instrucţiunea END
            Ok
Comanda:    CONT
Răspuns:    Sfîrşitul programului
            Ok
```

Observaţie. Secvenţa 10 la 70 prezentată mai sus cuprinde de fapt trei fraze (programe) ce pot fi apelate şi executate independent.

De exemplu, dacă se introduce comanda

```
RUN 40
```

se obţine:

```
Urmează instrucţiunea END
Ok
```

În concluzie o **frază BASIC** poate conţine una sau mai multe **propoziţii** separate prin semnul de punctuaţie `':'` sau caracterul `<CR>`. Sfîrşitul frazei este semnalat printr-una din instrucţiunile `STOP` sau `END`. Funcţie de problema de soluţionat ea poate fi mai mult sau mai puţin complexă şi, de regulă, conţine secvenţe de dialog cu utilizatorul.

3.4 Vocabular şi elemente de gramatică

Principalele elemente de vocabular ale limbajului BASIC sînt: **setul de caractere, constantele, cuvintele rezervate** (cheie) **identificatorii şi variabilele.**

Gramatica limbajului o constituie mulţimea regulilor semantice şi sintactice de formare a propoziţiilor corecte BASIC (instrucţiuni, expresii) precum şi a frazei (programului).

● Setul de caractere

Reprezintă totalitatea caracterelor ce pot fi utilizate pentru scrierea programelor BASIC. Setul de caractere cuprinde caractere alfabetice (literele mari şi mici ale alfabetului latin), caractere numerice (cifrele zecimale 0 la 9) şi caractere speciale `( + = - * \ / ^ () % # $ ! , ; : ' . & @ _ " <RUBOUT> <ESC> <LF> <CR> )`.

În BASIC, literele mari şi mici, cu excepţia celor ce apar în constante şir sau nume de fişier, sînt echivalente.

● Constante

Sînt mărimi introduse direct în program şi ale căror valori rămîn neschimbate pe toată durata execuţiei acestuia. În BASIC există două tipuri de constante: numerice şi şiruri.

Constantele numerice reprezintă valori numerice reale ce pot fi exprimate, funcţie de tipul lor, sub următoarele forme:

- întregi, cuprinse între -2^{15} şi $2^{15}-1$ (`-32768` şi `32767`);
- neîntregi cu punct zecimal: `213.46`;
- neîntregi cu exponent: `32E7`;
- în dublă precizie: `65589D12`;
- hexazecimale (precedate de `&H`): `&H2F4A`;
- octale (precedate de `&O`): `&O1777`.

Reprezentarea internă a constantelor se poate efectua în două moduri:

- în simplă precizie (dacă are maxim 7 cifre zecimale, este exprimată în forma cu exponentul `'E'` sau se termină cu caracterul `'!'`);
- în dublă precizie (dacă are 8 sau mai multe cifre zecimale, este exprimată în forma cu exponentul `'D'` sau se termină cu caracterul `'#'`).

Constantele `'sir'` sînt secvenţe de maxim 255 caractere alfanumerice cuprinse între ghilimele.

● Cuvinte rezervate

Sînt nume simbolice cu semnificaţii particulare în semantica limbajului. Numărul cuvintelor cheie şi semnificaţiile lor concrete depind de varianta de BASIC utilizată. În principal cuvintele cheie cuprind numele comenzilor, ale funcţiilor intrinseci precum şi cuvintele utilizate în construcţia instrucţiunilor.

Identificatori

Sînt cuvinte utilizate într-un program ca nume de dată şi permit descriera simbolică a acţiunilor în care acestea sînt implicate. Identificatorii mai poartă şi denumirea de 'nume simbolice'.

Un identificator începe obligatoriu cu o literă ce poate fi urmată de unul sau mai multe caractere alfabetice, numerice sau de caracterul '.' şi trebuie să nu coincidă cu un cuvînt rezervat.

Exemple: `A  A1R  FAM.2  CASA`

Observaţie. *Dacă un identificator începe cu literele* `'FN'` *atunci el este nume de funcţie definită de utilizator.*

Variabile

Variabilele constituie entităţi indivizibile în raport cu prelucrările specificate în BASIC. Spre deosebire de constante, valorile variabilelor pot fi modificate pe parcursul execuţiei programului ce le conţine.

Într-un program fiecărei variabile îi corespunde un identifiator utilizat în descrierea simbolică a acţiunilor în care acestea apar astfel încît, în reprezentarea operaţiilor efectuate cu variabile se utilizează nu valorile, ci numele lor. De exemplu propoziţia

```
A = B + C
```

are semnificaţia: se atribuie variabilei `'A'` valoarea sumei variabilelor `'B'` şi `'C'`.

Limbajul BASIC permite lucrul cu variabile numerice (simple sau indexate) şi variabile şir.

Varabilele numerice pot fi: întregi, simplă precizie sau dublă precizie, tipul acestora fiind definit printr-un caracter special ce urmează numelui variabilei, astfel:

`'%'` variabilă întreagă;
`'!'` variabilă simplă precizie;
`'#'` variabilă dublă precizie.

Exemple de variabile numerice simple: `A!  SA%  B.C!  DRF#.`

Numele unei variabile indexate este format dintr-un identificator acceptat de BASIC (eventual însoţit de caracterul special de tip), urmat de una sau mai multe expresii întregi între paranteze rotunde.

Exemple: `VEC(6)  CF#(I,J)  MAT%(K,L+1).`

Spre deosebire de variabilele simple, variabilele indexate trebuie declarate înainte de utilizarea cu instrucţiunea neexecutabilă `DIM`, avînd sintaxa:

$$\texttt{DIM ident}_1\ (n_{11}\ [,n_{12}\ \ldots])[\texttt{ident}_2\ (n_{21}\ [,n_{22}\ \ldots])\ldots]$$

unde

`ident` reprezintă identificatorul variabilei;
n_{ij} reprezintă valorile maxime ale indicilor.

Instrucţiunea DIM alocă spaţiul necesar tablourilor (variabilelor indexate) şi iniţializează toate componentele cu 0.

Referirea unui element de tablou se efectuează prin identificator şi indicii corespunzători poziţiei sale în cadrul acestuia (vezi capitolul 2.4).

Dacă de la un moment dat, pe parcursul execuţiei unui program, unele variabile indexate nu mai sînt necesare, pentru eliberarea spaţiului alocat şi a numelor rezervate variabilelor, se poate utiliza instrucţiunea ERASE, cu sintaxa:

ERASE $ident_1$ [,$ident_2$ ...]

unde $ident_i$ sînt identificatori de variabile indexate ce figurează într-o instrucţiune DIM anterioară. Dacă $ident_i$ nu figurează într-o instrucţiune DIM anterioară sau a fost eliberată prin alt ERASE se semnalează eroare.

Indicii unei variabile indexate iau valori întregi. Valoarea minimă implicită este 0. Această valoare poate fi modificată cu instrucţiunea:

OPTION BASE exp

care stabileşte valoarea minimă a indicelui de tablouri la valoarea expresiei numerice întregi exp, ce poate fi 0 sau 1.

Variabilele de tip şir sînt semnalate prin prezenţa caracterului '$' imediat după identificatorii acestora.

Exemple de variabile şir: AD$ VECT$ SIR1.A$.

Remarcă. *Dacă un identificator nu este urmat de un caracter special care să-i definească tipul, atunci variabila respectivă este considerată variabilă numerică simplă precizie.*

Observaţie. *Utilizarea caracterelor speciale pentru definirea tipurilor variabilelor oferă posibilitatea utilizării aceluiaşi identificator pentru variabile diferite. Astfel variabilele*

A% A A# A(I) A%(I) A#(I) A$ A$(I)

deşi au acelaşi identificator (A) sînt distincte.

Pe lîngă caracterele speciale, tipurile variabilelor BASIC pot fi definite şi după prima literă din identificator prin utilizarea instrucţiunilor DEFINT, DEFSNG, DEFDBL şi DEFSTR care au sintaxa:

$$\text{DEF} \begin{cases} \text{INT} \\ \text{SNG} \\ \text{DBL} \\ \text{STR} \end{cases} l_1 \ [,l_2 \ \ldots]$$

unde 'l_1 ,l_2 ,...' reprezintă o listă de litere.

Tipul variabilelor ai căror identificatori încep cu literele specificate în listă este dat de sufixul utilizat, astfel:

INT	variabilă întreagă;
SNG	variabilă simplă precizie;
DBL	variabilă dublă precizie;
STR	variabilă şir.

Iniţializarea variabilelor numerice cu 0 şi a variabilelor şir cu caracterul "" (şir nul), se poate efectua global, prin comanda CLEAR cu sintaxa:

```
CLEAR [,mem][,stiva].
```

Utilizată fără parametrii, comanda CLEAR, determină iniţializarea cu 0 a tuturor variabilelor numerice, cu caracterul "" (NUL) a celor şir, precum şi închiderea tuturor fişierelor deschise şi eliberarea zonelor tampon ataşate acestora.

Utilizată cu parametrii, comanda CLEAR, fixează limita memoriei disponibile pentru interpretor la adresa 'mem' şi / sau rezervă pentru stivă, 'stiva' octeţi.

● Operatori

Sînt simboluri care desemnează funcţii de calcul aritmetic, logic sau de comparaţie a valorilor uneia sau mai multor variabile, pe baza unor date asupra cărora se efectuează operaţii.

În BASIC există operatori aritmetici, relaţionali, logici (prezentaţi, în ordinea descrescătoare a priorităţilor, în tabelul 3.1) şi funcţionali. Ordinea de execuţie a operaţiilor poate fi modificată prin utilizarea parantezelor rotunde '(' şi ')'.

OPERATORI BASIC

Operator[1)]	Semnificaţie	Categorie
^	ridicare la putere	Aritmetici
-	negaţie (înmulţire cu -1)	
* /	înmulţire, împărţire	
\ MOD	înmulţire întreagă, restul împărţirii întregi	
+ -	adunare, scădere	
< >	mai mic, mai mare	Relaţionali
=	egal	
=> (>=)	mai mare sau egal	
<= (=<)	mai mic sau egal	
<> (><)	diferit	

NOT	negaţie logică	Logici[2]
AND	SI logic	
OR	SAU logic	
XOR	sau exclusiv	
IMP	implicaţie (`X IMP Y = 1 daca X <= Y`)	
EQV	echivalenţă (`X EQV Y = 1 daca X = Y`)	

[1] Operatorii de aceeaşi prioritate determină execuţia operaţiilor de la stînga la dreapta
[2] Acţionează la nivel de bit

Tabelul 3.1

Operatorii aritmetici `'\'` şi `'MOD'` se pot utiliza şi cu operanzi neîntregi, dar, în acest caz, înaintea efectuării operaţiei, aceştia sînt convertiţi la întregi.

Operatorii relaţionali permit compararea a două valori, rezultatul operaţiei fiind **adevărat** (valoare numerică diferită de zero) sau **fals** (valoare numerică zero).

Operatorii logici permit efectuarea operaţiilor logice la nivel de bit (vezi capitolul 3.6).

Operatorii funcţionali sînt nume de funcţii intrinseci (predefinite: `SQR` - radical -, `LOG` - logaritm natural -, etc.) sau de funcţii definite de utilizator.

● Expresii

Sînt succesiuni de constante şi variabile legate prin operatori şi, eventual, cuprinse în paranteze.

Funcţie de operatorii utilizaţi, ele pot fi: **aritmetice** (caz în care rezultatul este o valoare numerică) şi **logice** (rezultatul fiind o valoare de tip boolean: `0 = fals` şi `<> 0 = adevarat`).

Remarcă. *Rezultatul evaluării unei expresii logice poate fi folosit şi ca atare (el fiind în fond tot o valoare numerică); a se vedea în acest sens capitolul 3.6.*

Dacă într-o expresie intervin variabile sau constante de tipuri diferite, atunci înaintea efectuării operaţiilor, operanzii sînt convertiţi astfel:

- operatorii aritmetici convertesc toţi operanzii la tipul operandului cu precizia cea mai mare, care este şi tipul rezultatului expresiei;
- operanzii logici convertesc operatorii în întregi şi produc un rezultat întreg.

În cazul în care valoarea expresiei este atribuită unei variabile, atunci ea este convertită la tipul acesteia.

Observaţii

1. *La conversia din virgulă mobilă în întreg, se rotunjeşte la întregul cel mai apropiat.*
2. *La conversia din dublă precizie în simplă precizie se reţin primele 7 cifre zecimale, ultima fiind obţinută prin rotunjire.*

● Comentarii

Pentru a uşura parcurgerea textului şi înţelegerea unui program, în textul sursă se pot insera diferite comentarii.

Introducerea comentariilor în textul sursă se poate efectua în două moduri.

O primă modalitate constă în utilizarea instrucţiunii `REM` cu sintaxa:

```
REM comentariu.
```

Aceasta poate figura oriunde într-un program.

O altă modalitate de introducere a comentariilor este precedarea acestora de caracterul apostrof (`'`). Comentariile nu sînt luate în considerare la execuţia programelor ce le conţin.

Atenţie. Dacă se lucrează cu mai multe instrucţiuni pe o linie BASIC, atunci instrucţiunea REM, respectiv comentariul introdus după caracterul apostrof, trebuie să fie ultimele (să preceadă caracterul <CR>), în caz contrar toate instrucţiunile sau comenzile aflate între acesta şi sfîrşitul liniei (marcat prin caracterul <CR>) sînt considerate comentarii şi ca atare, la execuţie nu sînt luate în considerare.

3.5 Instrucţiuni de intrare / ieşire

Instrucţiunile de intrare / ieşire permit atribuirea de valori unor variabile utilizînd o listă de constante sau de la tastatură precum şi afişarea de valori la display.

Generarea listelor de constante se efectuează cu instrucţiunea neexecutabilă

```
DATA const[,const,...].
```

Ea permite memorarea de constante numerice şi şiruri de caractere ce pot fi apoi citite cu instrucţiunea `READ`. Constantele numerice trebuie separate prin virgulă, iar şirurile de caractere trebuie încadrate în ghilimele (`"`). Instrucţiunile `DATA` pot fi plasate oriunde în program. Valorile din lista de constante sînt memorate într-un singur bloc, astfel încît, în cazul cînd sînt mai multe instrucţiuni `DATA` într-un program, listele lor se concatenează, iar citirea se efeçtuează în ordinea în care apar faţă de prima constantă din prima instrucţiune `DATA` apărută în program.

Instrucţiunea `READ` are sintaxa

```
READ var[,var,...]
```

şi permite atribuirea de valori din lista creată cu instrucţiunea (instrucţiunile) `DATA`, variabilelor (`'var'`) specificate.

Instrucţiunea `READ` începe atribuirile cu prima constantă ce urmează celor atribuite printr-un `READ` anterior, nefiind necesară concordanţa între numărul instrucţiunilor `DATA` şi `READ`. Concordanţa se impune însă între tipurile constantelor şi al variabilelor cărora le sînt atribuite. Dacă numărul elementelor variabilelor specificate depăşeşte numărul constantelor se semnalează eroare, iar în caz

că numărul lor este mai mic, constantele necitite sînt ignorate.

De exemplu, în urma executării secvenţei

```
15   DIM D(2)
20   DATA 5,7.2,"ABC",3,2
      .
50   READ A%,B,C$
      .
80   DATA 1,3,6.3
      .
100  READ D(0),D(1)
      .
105  READ E%
```

se vor efectua următoarele atribuiri:

```
A% = 5          B = 7.2          C$ = "ABC"
D(0) = 3        D(1) = 2         E%=1
```

Constantele 3 şi 6.3, din lista instrucţiunii DATA de la linia 80 vor fi ignorate.

Exemplul 3.3

Sub rezerva descrierii ulterioare a instrucţiunii PRINT, în listingul 3.1 se exemplifică modul de citire a constantelor dintr-o listă DATA.

```
10   'Citirea constantelor din lista si
15   'afisarea lor la terminal
20   DATA 3.24,7.5,"SIR",8153,0.01,8
25   READ A,B,C$
30   READ D,E,F
35   PRINT "A =",A;"B =",B;"C =",C$
40   PRINT "D =",D;"E =",E;"F =",F
50   STOP
RUN
A =          3.24 B =          7.5 C =          SIR
D =          8153 E =          .01 F =            8
Break in 50
Ok
```

Listing 3.1

Pentru a facilita citirea repetată a unui bloc de dată se utilizează instrucţiunea

```
RESTORE.
```

Această determină repoziţionarea la începutul blocului de date, astfel încît, primul READ ce urmează unui RESTORE, va începe citirea (atribuirea) de la prima constanta din bloc.

De exemplu secvenţa

```
10   DATA 1,5.1,6
20   READ A,B
30   RESTORE
40   READ C,D,E
```

determină atribuirile:

```
A = 1          B = 5.1
C = 1          D = 5.1        E = 6 .
```

Citirea (introducerea) datelor de la tastatură se realizează cu instrucţiunea

```
INPUT[;] ["text",] var1 [,var2 ...]
```

Execuţia acesteia provoacă întreruperea programului şi apariţia pe ecran a mesajului `'text'`, dacă acesta a fost specificat. Datele introduse sînt atribuite în ordinea variabilelor din listă. Între tipul variabilelor şi valorile introduse trebuie să fie o corespondenţă biunivocă. De asemenea trebuie să existe concordanţă între numărul elementelor variabilelor din listă şi valorile introduse. În caz contrar, se semnalează eroare, iar atribuirile nu se efectuează pînă la introducerea corectă.

***Recomandare.** Pentru a reduce la minim posibilitatea introducerii de date eronate este bine ca fiecare instrucţiune `INPUT` să fie însoţită de `'text'` şi să se solicite introducerea unei singure valori.*

De exemplu, în locul secvenţei:

```
30   INPUT "zi,luna,an :",ZI,LUNA,AN
```

se recomandă

```
30   INPUT "zi    :",ZI
35   INPUT "luna  :",LUNA
40   INPUT "an    :",AN
```

Pentru introducerea de la consolă a unei linii (de maxim 255 caractere) se poate utiliza instrucţiunea

```
LINE INPUT[;] ["text" ] vars
```

În urma execuţiei acesteia, textul introdus (terminat cu caracterul <CR>) este depus în variabila şir `'vars'`. `'text'` are semnificaţia prezentată la instrucţiunea `INPUT`.

Pentru afişarea la display sau la imprimantă a valorilor variabilelor sau a unor anumit situaţii de ieşire limbajul BASIC dispune de instrucţiunile `PRINT` şi `PRINT USING`, respectiv `LPRINT` şi `LPRINT USING`.

Instrucţiunea `PRINT` are sintaxa

```
PRINT [  ["text"]  [exp][  ["text"]     [exp]...     ]
```

şi permite afişarea pe eran a valorilor expresiilor din listă într-un format dependent de caracterele utilizate pentru separarea elementelor în listă astfel:

`','` - afişarea valorilor următoare se efectuează cîte una pe zonă (o zonă = 14 poziţii); dacă o variabilă de tipărit depăşeşte lungimea unei zone, elementul următor este tipărit la începutul primei zone libere;

`';'` sau `' '`
- afişarea valorii următoare se efectuează imediat după cea anterioară.

Observaţie. La afişare, valorile numerice sînt urmate de un spaţiu.

Instrucţiunea `PRINT USING` are sintaxa:

```
PRINT USING sirf;exp1 [,exp2 ...]
```

şi permite afişarea valorilor unei liste de expresii numerice sau şir de caractere, conform formatului specificat în `'sirf'`. Şirul de formate `'sirf'` poate fi un şir de literale anterior definit sau şir de variabile, specificînd formatele cîmpurilor în care urmează a se tipări valorile expresiilor din listă.

Pentru specificarea modului de afişare a cîmpurilor se utilizează unul sau mai multe caractere speciale specifice afişării variabilelor şir, respectiv celor numerice.

Caracterele utilizate în afişarea şirurilor sînt:

`'!'` - afişează numai primul caracter din şir;

`'\n_spatii\'`
- se afişează `n+2` caractere din şir; dacă şirul este mai lung, ultimele caractere se neglijează iar dacă este mai scurt, se completează cu zero;

`'&'` - cîmp rezervat pentru afişarea unui şir de lungime variabilă; şirul este afişat în întregime;

Exemplul 3.4

Programul prezentat în listingul 3.2 permite introducerea de la tastatură a două şiruri de caractere `'SIR1'` şi `'SIR2'` şi reliefează diversele modalităţi de afişare ale acestora.

```
10    'Citirea si scrierea sirurilor de caractere
15    INPUT "SIR1 = ",A$
20    INPUT "SIR2 = ",B$
25    PRINT "SIR1 = ",A$," SIR2 = ",B$
30    PRINT "SIR1 = ";A$;" SIR2 = ";B$
35    PRINT USING "!";A$;" !";B$
40    PRINT USING " \      \";A$;B$
45    PRINT USING "&";A$;" ";B$
50    END
RUN
SIR1 = ACESTA_ESTE_SIRUL_UNU
SIR2 = SIRUL_DOI
SIR1 =       ACESTA_ESTE_SIRUL_UNU        SIR2 =       SIRUL_DOI
SIR1 = ACESTA_ESTE_SIRUL_UNU SIR2 = SIRUL_DOI
A S
 ACESTA_ SIRUL_D
ACESTA_ESTE_SIRUL_UNU SIRUL_DOI
Ok
```

Listing 3.2

Caractere utilizate în afişarea numerelor sînt:

'#' - reprezintă poziţia fiecărei cifre a numărului;
'+' - aflat la începutul sau la sfîrşitul şirului format, determină afişarea semnelor numerelor (+ sau -) înainte, respectiv în urma lor;
'-' - pus la sfîrşitul formatului, determină afişarea semnului pentru numerele negative;
'*' - puse la începutul formatului determină apariţia de asteriscuri înaintea unor numerice şi oferă posibilitatea afişării unor numere cu două cifre mai lungi decît formatul specificat ;
'$$' - determină apariţia unui '$' la stînga numărului şi creează posibilitatea afişării unei cifre în plus faţă de formatul specificat ;
'**$' - combină efectele caracterelor '**' şi '$$';
',' - plasat la stînga punctului zecimal, determină apariţia unei virgule la fiecare trei cifre în stînga acestuia; plasat înaintea formatului determină apariţia unei virgule după fiecare număr afişat;

Observaţie. *Dacă se utilizează semnul exponenţial '^^^^', virgula nu are efect; '^^^^' plasat după poziţiile cifrelor determină afişarea valorii în format exponenţial; ''' (apostrof) determină afişarea caracterului care îi urmează. Dacă numărul de afişat este mai mare decît cîmpul specificat prin format, atunci, înaintea numărului, este afişat caracterul '%'.*

Exemplul 3.5

În listingul 3.3 se prezintă un program (şi rezultatele rulării acestuia) care exemplifică efectul utilizării diferitelor caractere admise în descrierea modului de afişare al variabilelor numerice.

```
 5     PRINT "Citirea si scrierea variabilelor numerice"
10     INPUT "N1 = ",N1
15     INPUT "N2 = ",N2#
20     PRINT "N1,N2",N1,N2#
25     PRINT USING "      #####.### ";N1;N2#
30     PRINT USING "      #####.# ";N1;N2#
35     PRINT USING "     +#####.### ";N1;N2#
40     PRINT USING "      #####.###- ";N1;N2#
45     PRINT USING "   **######.###- ";N1;N2#
50     PRINT USING "   $$#####.###- ";N1;N2#
55     PRINT USING "  **$#####.###- ";N1;N2#
60     PRINT USING " **$#####,.###- ";N1;N2#
65     PRINT USING "~#####,.###- ";N1;N2#
70     PRINT USING "#####^^^^, ";N1;N2#
75     PRINT USING "#####.###, ";N1;N2#
80     END
RUN
Citirea si scrierea variabilelor numerice
N1 = 1234
N2 = 987654.321
N1,N2           1234           987654.321
      1234.000       %987654.321
      1234.0         %987654.321
     +1234.000       %+987654.321
      1234.000       %987654.321
   ***1234.000       *987654.321
     $1234.000       $987654.321
  ***$1234.000       *$987654.321
```

```
 ***$1,234.000    *$987,654.321
~ 1,234.000   ~%987,654.321
 1234E+00,   9877D+02,
 1234.000, %987654.321,
Ok
```

Listing 3.3

Pentru afişarea datelor la terminal se mai poate utiliza instrucţiunea:

WRITE [exp_1 [,exp_2 ...]]

Spre deosebire de PRINT, aceasta inserează automat <CR><LF> după ultimul element din listă.

Instrucţiunile

LPRINT şi LPRINT USING

determină afişarea valorilor variabilelor din listă la imprimantă. Au aceleaşi sintaxe cu instrucţiunile PRINT, respectiv PRINT USING.

Pentru stabilirea lungimii liniei la display sau imprimantă se utilizează instrucţiunea WIDTH, cu sintaxa:

WIDTH [LPRINT] exp

unde 'exp' reprezintă o expresie întreagă ce trebuie să aibă valori cuprinse între 0 şi 255 şi reprezintă numărul maxim de caractere admis pe o linie. Dacă această instrucţiune nu este prezentă, linia se consideră a fi de lungime implicită specifică dispozitivului respectiv.

Observaţie. *Limita inferioară a numărului de caractere, precum şi numărul implicit de caractere pe linie diferă funcţie de varianta de BASIC utilizată.*

Exemplul 3.6

Programul ilustrat în listingul 3.4 prezintă efectul utilizării instrucţiunii WIDTH în două cazuri (cu argument 23, respectiv 40) asupra unui şir de caractere preluat de la tastatură cu instrucţiunea LINE INPUT.

```
10    DATA "Linie de","caractere"
15    READ L$,C$
20    LINE INPUT "SIR = ",A$
24    N=23
25    WIDTH N
30    PRINT L$;N;C$
35    PRINT A$
39    N=40
40    WIDTH N
45    PRINT L$;N;C$
50    PRINT A$
55    WIDTH 80
60    END
RUN
SIR = Acesta_este_un_sir_mai_lung_de_patruzeci_de_caractere
```

```
Linie de 23 caractere
Acesta_este_un_sir_mai_
lung_de_patruzeci_de_ca
ractere
Linie de 40 caractere
Acesta_este_un_sir_mai_lung_de_patruzeci
_de_caractere
Ok
```

Listing 3.4

Observaţie. *Instrucţiunea* `'WIDTH 80'` *din linia 55 are rolul de a restabili lungimea liniei display la 80 caractere; altfel, după execuţia programului lungimea liniei ar fi rămas de 40 caractere.*

3.6 Operaţii aritmetice şi logice

Permit rezolvarea expresiilor aritmetice şi logice, indiferent de forma şi complexitatea acestora. Evaluarea expresiilor se efectuează într-o anumită ordine (dată de priorităţile operatorilor utilizaţi) iar rezultatul, de regulă se atribuie unei variabile. De remarcat faptul că, în urma evaluării unei expresii, ca şi în alte limbaje de programare, în BASIC trebuie să avem o singură valoare rezultat (o singură soluţie). În cazul cînd în rezolvarea unei probleme pot rezulta mai multe soluţii, acestea vor fi de aşa natură descrise în BASIC încît fiecare să fie determinată separat.

Instrucţiunea care permite atribuirea valorii rezultate în urma evaluării unei expresii la o variabilă este `LET`. Această instrucţiune are sintaxa:

```
[LET] var=exp
```

unde `var` este variabila care primeşte valoarea rezultată prin evaluarea expresiei `exp`. Cuvîntul cheie `LET` este opţional (poate lipsi). Din motive de simplitate şi eficienţă, în general, acest cuvînt nu se utilizează.

Într-o instrucţiune `LET` pot apare variabile numerice sau variabile şir (nu simultan).

Variabilele numerice pot fi utilizate în operaţii aritmetice şi logice, iar variabilele şir numai în atribuiri simple şi concatenări.

● Operaţii aritmetice

Adunarea şi scăderea se efectuează cu ajutorul operatorilor `'+'`, respectiv `'-'`, avînd semnificaţia cunoscută din matematică.

Înmulţirea şi împărţirea numerelor reale se efectuează utilizînd operatorii `'*'`, respectiv `'/'`.

Împărţirea întreagă a două numere întregi (numerele reale sînt automat convertite în întregi) se efectuează cu operatorul `'\'`. Evident, în cazul a două numere întregi `'A'` şi `'B'`, prime între ele,

```
C1 = A / B    diferă de  C2 = A \ B .
```

De exemplu, în urma execuţiei secvenţei:

```
 5   DATA 10,4
10   READ A,B
15   C1=A/B
20   C2=A\B
```

variabilei C1 i se va atribui valoarea '2.5', iar variabilei C2 valoarea '2'.

Restul împărţirii întregi a două numere se obţine utilizînd operatorul MOD. De exemplu:

```
10   DATA 19,3
20   READ A,B
30   C=A MOD B
```

după evaluarea expresiei din linia 30 variabilei C i se va atribui valoarea '1'.

Cu ajutorul operatorilor '\' şi 'MOD' putem alcătui o secvenţă care să descrie împărţirea cu rest:

```
50   CIT% = A% \ B%
60   REST% = A% MOD B%
```

(unde B% <> 0). Evident

```
A% = CIT% * B% + REST% .
```

Observaţie. La întocmirea şi descrierea în BASIC a expresiilor aritmetice trebuie avut în vedere că evaluarea lor să ducă la un număr finit (atenţie la împărţirea cu 0) şi reprezentabil în calculator. În caz contrar, la execuţie, va fi semnalată eroare.

Ridicarea la putere se efectuează cu operatorul '^'

Operaţiile aritmetice se efectuează în ordinea:

- ridicarea la putere (^);
- înmulţirea (*), împărţirea (/), împărţirea întreagă (\) şi restul împărţirii întregi (MOD);
- adunarea (+) şi scăderea (-).

Dacă este necesar, această ordine poate fi modificată utilizînd parantezele '(' şi ')'.

Exemplul 3.7

Să se scrie un program BASIC pentru calculul şi afişarea valorilor expresiei:

$$E(X,Y) = \frac{1}{2}\,\frac{X^{\frac{1}{3}}}{Y^{\frac{1}{4}}}\cdot\frac{X^{\frac{1}{2}}+Y^{2}}{X+Y}+\frac{5-X^{\frac{1}{2}}}{X-Y}$$

cu X > 0 , Y > 0 şi X <> Y .

Programul BASIC şi soluţia unei execuţii sînt prezentate în listingul 3.5.

```
10    PRINT "Program pentru calculul expresiei E(x,y)"
15    INPUT "Introduceti X (X>=0) : ",X
20    INPUT "Introduceti Y (Y>0 SI Y<>X) : ",Y
```

```
25    E=(X^(1/3)/Y^(1/4)*(X^(1/2)+Y^2)/(X+Y)+
        (5-X^(1/2))/(X-Y))/2
30    PRINT USING "######^^^^";E
35    END
RUN
Program pentru calculul expresiei E(x,y)
Introduceti X (X>=0) : 6
Introduceti Y (Y>0 SI Y<>X) : 4
 18229E-04
Ok
```

Listing 3.5

● Operaţii logice

Sînt operaţii care se efectuează bit cu bit. Funcţie de tipul variabilelor numerice implicate, numărul de biţi cu care se operează poate fi 8, 16, 32 sau 64.

Exemplul 3.8

Fie două numere întregi `N%` şi `L%`. Utilizînd pe rînd operatorii logici să determinăm valorile variabilei

```
M% = N% 'opl' L%
```

unde `opl` este un operator logic. Dacă luăm $N\%=26_{(10)} = 11010_{(2)}$ şi $L\%=47_{(10)} = 101111_{(2)}$, atunci pentru `M%` obţinem valorile din tabelul 3.2.

Programul BASIC şi rezultatele rulării cu valorile de mai sus sînt prezentate în listingul 3.6.

```
 5    PRINT "Operatii logice"
10    INPUT "N = ",N%
15    INPUT "L = ",L%
20    M%=NOT N%
25    PRINT "NOT N   =";M%
30    M%=N% AND L%
35    PRINT "N AND L =";M%
40    M%=N% OR L%
45    PRINT "N OR  L =";M%
50    M%=N% XOR L%
55    PRINT "N XOR L =";M%
60    M%=N% IMP L%
65    PRINT "N IMP L =";M%
70    M%=N% EQV L%
75    PRINT "N EQV L =";M%
80    END
```

```
RUN
Operatii logice
N = 26
L = 47
NOT N   =-27
N AND L = 10
N OR  L = 63
N XOR L = 53
N IMP L =-17
N EQV L =-54
Ok
```

Listing 3.6

EVALUAREA EXPRESIILOR LOGICE

Operaţie logică	Evaluare				Exemple[1] pentru: X=N%=$26_{(10)}$=$11010_{(2)}$ Y=L%=$47_{(10)}$=$101111_{(2)}$
	X	Y	Rezultat		
			Numeric	Logic[2]	
NOT X	1	-	0	A	X=0000000000011010 NOT X=$1111111111100101_{(2)}$=$-27_{(10)}$
	0	-	1	F	
X AND Y	1	1	1	F	X=0000000000011010 Y=0000000000101111 X AND Y=$0000000000001010_{(2)}$=$10_{(10)}$
	1	0	0	A	
	0	1	0	A	
	0	0	0	A	
X OR Y	1	1	1	F	X=0000000000011010 Y=0000000000101111 X OR Y=$0000000000111111_{(2)}$=$63_{(10)}$
	1	0	1	F	
	0	1	1	F	
	0	0	0	A	
X XOR Y	1	1	0	A	X=0000000000011010 Y=0000000000101111 X XOR Y=$0000000000110101_{(2)}$=$53_{(10)}$
	1	0	1	F	
	0	1	1	F	
	0	0	0	A	
X IMP Y	1	1	1	F	X=0000000000011010 Y=0000000000101111 X IMP Y=$1111111111101111_{(2)}$=$-17_{(10)}$
	1	0	0	A	
	0	1	1	F	
	0	0	1	F	
X EQV Y	1	1	1	F	X=0000000000011010 Y=0000000000101111 X EQV Y=$1111111111001010_{(2)}$=$-54_{(10)}$
	1	0	0	A	
	0	1	0	A	
	0	0	1	F	

[1] Vezi listingul 3.6; [2] A=Adevarat; F=Fals

Tabelul 3.2

● Operaţii cu variabile şir

Variabilele de tip şir pot apare într-o instrucţiune LET numai în două cazuri:
- în atribuiri simple de forma:

```
A$ = B$
```

- în concatenări (depunerea valorii unei variabile şir în continuarea valorii alteia) efectuate cu operatorul '+'.

Exemplul 3.9

Concatenarea a două şiruri (listingul 3.7).

```
10    PRINT "Concatenarea a doua siruri"
15    INPUT "Sirul A : ",A$
20    INPUT "Sirul B : ",B$
25    C$=A$+B$
30    PRINT "Sirul C este : ";
35    PRINT USING "&";C$
40    END
RUN
Concatenarea a doua siruri
Sirul A : concate
Sirul B : nare
Sirul C este : concatenare
Ok
```

Listing 3.7

Într-o instrucţiune LET pe lîngă operatorii amintiţi (aritmetici şi logici, în cazul variabilelor numerice, respectiv operatorul '+' <concatenare> în cazul variabilelor şir) mai pot apare şi operatori funcţionali, desemnînd funcţii numerice sau şiruri de caractere.

3.7 Funcţii şi subrutine

Funcţiile şi subrutinele sînt seturi de instrucţiuni generalizate, destinate îndeplinirii unei anumite acţiuni, apelabile din diferite puncte ale unui program.

Funcţiile sînt seturi de instrucţiuni apelabile printr-o mnemonică (un nume) şi care dau posibilitatea realizării directe a unor operaţii. Principala caracteristică a funcţiilor este aceea ca în urma evaluării expresiei (expresiilor) pe care o conţin, în program este returnată o singură valoare prin însăşi numele ei, astfel încît mnemonicele (identificatorii) lor apar direct în expresii aritmetice sau logice.

Subrutinele constituie module de program care permit realizarea uneia sau mai multor funcţiuni din cele mai diverse. Utilizarea subrutinelor permite realizarea de programe modulare uşor de proiectat şi depanat.

Observaţie. Majoritatea covîrşitoare a versiunilor BASIC nu dispune de subrutine (proceduri) în sensul celor întîlnite în alte limbaje de nivel înalt (FORTRAN, PASCAL, C, etc.), ele putînd fi însă, aşa cum vom vedea, simulate.

● Funcţii predefinite

În această categorie intră o serie de funcţii de utilizare frecventă cum sînt funcţiile matematice uzuale: exponenţială, logaritmică, valoare absolută, rădăcina pătrată, funcţii trigonometrice directe şi inverse etc., de intrare / ieşire, funcţii referitoare la şiruri, etc.

Principalele funcţii predefinite, tipul, semnificaţia şi modul de apel sînt prezentate în tabelul 3.3.

FUNCŢII PREDEFINITE (INTRINSECI)

Funcţie	Tip[1]		Rezultat (acţiune)	BA [2]	B80 [3]	GW [4]
	Funcţie	Argument				
ABS(X)	R	R	Valoarea absolută a lui X	*	*	*
ASC(X$)	I	S	Codul ASCII al primului caracter din şirul X$	*	*	*
ATN(X)	R	R	arctg(X) (în radiani -π/2<ATN(X)<π/2)	*	*	*
CDBL(X)	R	R	Conversie simplă, dublă precizie		*	*
CHR$(X%)	S	I	Returnează caracterul cu codul ASCII X%	*	*	*
CINT(X)	I	R	Conversie real, întreg	*	*	*
COS(X)	R	R	cos(X), (X în radiani)	*	*	*
CREAL(X%)	R	I	Conversie întreg, real	*		
CSNG(X)	R	R	Converteşte pe X în simplă precizie		*	*
CSRLIN	I	-	Returnează numărul liniei pe care se află cursorul			*
CVI(X$)	I	S	Converteşte un şir de 2 caractere numerice X$ în valoare întreagă		*	*
CVD(X$)	R	S	Converteşte un şir de 8 caractere numerice X$ în dublă precizie		*	*
CVS(X$)	R	S	Converteşte un şir de 4 caractere numerice X$ în simplă precizie		*	*
DATE$	S	-	Returnează un şir de 10 caractere conţinînd data zilei: ll-zz-aaaa			*
DEC$(X,F$)		R,S	Dă o reprezentare zecimală a lui X conform formatului F$ (vezi PRINT USING)	*		

DERR	I	-	Ultimul cod de eroare transmis de sistemul de gestiune al dischetei	*		
EOF(N%)	I	I	Returnează valoarea -1 dacă s-a detectat sfîrşitul fişierului de nr. logic N%	*	*	*
ERDEV	I	-	Codul de eroare depistat de ultimul dospozitiv ce a semnalat o eroare			*
ERDEV$	S	-	Şir de 2 sau 8 caractere conţinînd numele blocului, respectiv numele dispozitivului caracter ce a semnalat eroarea			*
ERL	I	-	Numărul liniei în care s-a depistat eroarea	*	*	*
ERR	I	-	Codul erorii depistate	*	*	*
EXP(X)	R	R	e^x	*	*	*
FIX(X)	I	R	Partea întreagă a lui X	*	*	*
FRE(0)	I	-	Numărul octeţilor nefolosiţi	*	*	*
HEX$(X%)	S	I	Şir reprezentînd valoarea hexazecimală a lui X%	*	*	*
HIMEM	I	-	Adresa maximă de memorie utilizată de BASIC	*		
INKEY$	S	-	Preia un caracter sau un şir nul de la tastatură	*	*	*
INP(X%)	I	I	Returnează octetul citit de la portul X%	*	*	*
INPUT$ (X%,#Y%)	S	I	Citeşte X% caractere de la terminalul sau dispozitivul Y%		*	*
INSTR (X%,Y$,Z$)	I	I,S	Caută prima apariţie a şirului Z$ în şirul Y$ începînd eventual de la poziţia X%	*	*	*
INT(X)	I	R	Returnează partea întreagă a lui X		*	*
IOCTL$ (#X%)	S	I	Şir de date de control privind modulul de control periferic relativ la dispozitivul de tip caracter cu nr. logic X%			*
JOY(X%)	I	I	Citeşte starea manetei de joc specificată prin X%	*		
KEY ON			Afişează pe ultima linie a ecranului primele 6 caractere ale valorilor tuturor celor 10 chei			*

LEFT$ (X$,Y%)	S	I	Returnează un subşir conţinînd Y% caractere din şirul X$	*	*	*
LEN(X$)	I	S	Returnează lungimea în octeţi a lui X$	*	*	*
LOC(X%)	I	I	Returnează poziţia curentă în fişierul cu numărul logic X%		*	*
LOF(X%)	I	I	Returnează lungimea în octeţi a fişierului cu numărul logic X%			*
LOG(X)	R	R	ln(X)	*	*	*
LOG10(X)	R	R	lg(X)	*		
LPOS(X%)	I	I	Returnează poziţia caracterului curent din zona tampon a imprimantei		*	*
LOWER$(X$)	S	S	Schimbă majusculele din X$ în litere mici	*		
MAX(list)	R	R	Determină valoarea maximă din lista	*		
MID$ (X$,Y%,Z%)	S	I,S	Returnează un subşir de Z% caractere din X$, începînd cu poziţia Y%	*	*	*
MIN(list)	R	R	Determină valoarea minimă din lista	*		
MKD$(X)	S	R	Conversie dublă precizie, şir de 8 octeţi		*	*
MKI$(X%)	S	I	Conversie întreg, şir de 2 octeţi		*	*
MKS$(X)	S	R	Conversie simplă precizie, şir de 4 octeţi		*	*
OCT$(X)	S	R	Returnează valoarea octală a lui X		*	*
PEEK(X%)	I*	I	Returnează valoarea întreagă a octetului citit din memorie, locaţia X%	*	*	*
PI	R	-	π	*		
PLAY	I	-	Returnează numărul de note din bufferul melodiei de fond			*
POS(X%)	I	I	Returnează poziţia cursorului pe linie	*	*	*
REMAIN(X%)	I	I	Returnează timpul care rămîne de măsurat cu cronometrul X% (0-3) înainte de a-l dezactiva	*		
RIGHT$ (X$,Y%)	S	I	Returnează un subşir a lui X$ conţinînd caracterele de la 1 la Y%	*	*	*

RND(X)	R	R	Returnează o valoare pseudoaleatoare în intervalul [0,1]; generatorul de numere aleatoare se inițializează cu instrucțiunea RANDOMIZE	*	*	*
ROUND (X,Y%)	R	R	Rotunjește X la Y% zecimale	*		
SGN(X)	I	R	Returnează: 1 dacă X>0; 0 dacă X=0; -1 dacă X<0.	*	*	*
SIN(X)	R	R	sin(X); X este exprimat în radiani	*	*	*
SPACE$(X%)	S	I	Returnează un șir de X% spații	*	*	*
SPC(X%)	-	I	Produce X% spații în linia de afișat	*	*	*
SQ(X%)	I	I	Indică starea firului de așteptare dintr-un canal sonor dat	*		
SQR(X)	R	R	Radical de ordinul 2 din X	*	*	*
STR$(X)	S	R	Returnează un șir de cifre corespunzător valorii lui X	*	*	*
STRING$ (X%,Y$)	S	I,S	Returnează un șir de X% caractere de codul primului caracter din Y$	*	*	*
TAB(X%)	-	I	Produce spații în linia de afișat până în poziția X%	*	*	*
TIME$	S	-	Returnează un șir de caractere conținînd ora sub forma: hh:mm:ss:			*
TAN(X)	R	R	tg(X); X este exprimat în radiani	*	*	*
USRn(X)	-	R	Activează o funcție scrisă de utilizator în limbaj mașină și îi transmite argumentul X; n∈[0,9] și este cel definit în DEF USR	*	*	*
UPPER$(X$)	S	S	Înlocuiește cu majuscule caracterele alfabetice din X$	*		
VAL(X$)	R	S	Returnează valoarea numerică a șirului de cifre X$	*	*	*
VARPTR(X)	I	-	Returnează adresa relativă a variabilei X sau a unei zone tampon disc		*	*

VARPTR$(X)	S	-	Returnează adresa relativă a variabilei X sau a unei zone tampon disc, sub forma unui şir de 3 caractere		*	*
XPOS	I	-	Returnează poziţia pe axa X a cursorului grafic	*		
YPOS	I	-	Returnează poziţia pe axa Y a cursorului grafic	*		

[1] I=Întreg, R=Real, S=Şir [2] BA=BASIC-AMSTRAD [3] B80=BASIC-80 (MBASIC, GBASIC)
[4] GW=GW-BASIC

Tabelul 3.3

Aşa cum s-a prezentat în capitolul 3.4 funcţiile constituie operatori şi deci apar direct în descrierea în BASIC a unei expresii.

Exemplul 3.10

În listingul 3.8 se prezintă o variantă de program pentru evaluarea expresiei:

$$E(X) = \left[\frac{\sqrt{X}}{3} \cdot \left(\sin(Y \cdot \frac{\pi}{180}) + \ln(X)\right)\right]$$

cu X > 0.

```
10     PRINT "Calculul expresiei E(x,y)"
15     INPUT "X (X>0) : ",X
20     INPUT "Y : ",Y
25     E=INT(SQR(X)/3*(SIN(Y*3.1415/180)+LOG(X)))
30     PRINT "E =";E
35     STOP
RUN
Calculul expresiei E(x,y)
X (X>0) : 32.1
Y : 270
E = 4
Break in 35
Ok
```

Listing 3.8

● Funcţii definite de utilizator

Pe lîngă funcţiile predefinite, în programele sale, utilizatorul are posibilitatea de a-şi defini propriile funcţii.

Pentru aceasta are la dispoziţie instrucţiunea DEF, cu sintaxa:

```
DEF FNa [(var1 [,var2 ...])] = exp
```

unde `FNa` reprezintă numele funcţiei, în care primele două litere sînt obligatoriu `FN` iar a trebuie să respecte convenţiile unui nume de variabilă; var_i reprezintă argumentele funcţiei; `exp` orice expresie BASIC admisă (incluzînd chiar şi funcţii anterior definite).

O funcţie poate fi de tip numeric sau şir şi este limitată la expresii ce pot fi scrise pe cel mult o linie.

Definirea unei funcţii trebuie efectuată înaintea primei ei utilizări.

Exemplul 3.11

Să se definească o funcţie care să permită determinarea suprafeţei unui hexagon funcţie de raza cercului circumscris acestuia şi să se determine cu ajutorul ei suma suprafeţelor a două hexagoane. Dacă notăm cu `R` raza cercului circumscris, atunci suprafaţa este

$$S = R^2 * \frac{\sqrt{3}}{2}$$

Programul şi soluţia obţinută sînt prezentate în listingul 3.9.

```
10    'Definirea si utilizarea unei functii"
15    DEF FNSH(R)=R^2*SQR(3)/2
20    INPUT "R1 (R1>0) = ",R1
25    INPUT "R2 (R2>0) = ",R2
30    S=FNSH(R1)+FNSH(R2)
35    PRINT "S1 =";FNSH(R1);"S2 =";FNSH(R2);"S =";S
40    STOP
RUN
R1 (R1>0) = 4
R2 (R2>0) = 6
S1 = 13.8564 S2 = 31.1769 S = 45.0333
Break in 40
Ok
```

Listing 3.9

● Subrutine BASIC

În general, subrutinele sînt utilizate în cazul cînd o secvenţă de instrucţiuni este folosită de mai multe ori într-un program, pentru a elimina necesitatea repetării acestora.

O utilizare însă mult mai eficientă a subrutinelor se obţine atunci cînd, pentru o clasă de probleme se construiesc (sau sînt constituite) **biblioteci** de subrutine. În acest caz, realizarea programelor (aşa cum se va vedea în capitolele următoare) se poate simplifica într-atît încît să se ajungă ca acesta să fie alcătuit doar din apeluri la subrutine deja definite şi testate.

În lucrul cu subrutine se utilizează două instrucţiuni: `GOSUB` şi `RETURN`.

Instrucţiunea `GOSUB` are sintaxa:

```
GOSUB etich
```

unde `etich` reprezintă adresa primei linii a subrutinei.

În momentul întîlnirii acestei instrucţiuni, programul este întrerupt, se reţine adresa următoarei

instrucţiuni şi se face salt la linia cu numărul `etich`, execuţia continuînd cu prima instrucţiune executabilă de după aceasta pînă în momentul întîlnirii instrucţiunii RETURN.

Instrucţiunea RETURN are sintaxa:

RETURN

şi determină revenirea la adresa instrucţiunii imediat următoare ultimei instrucţiuni GOSUB executate.

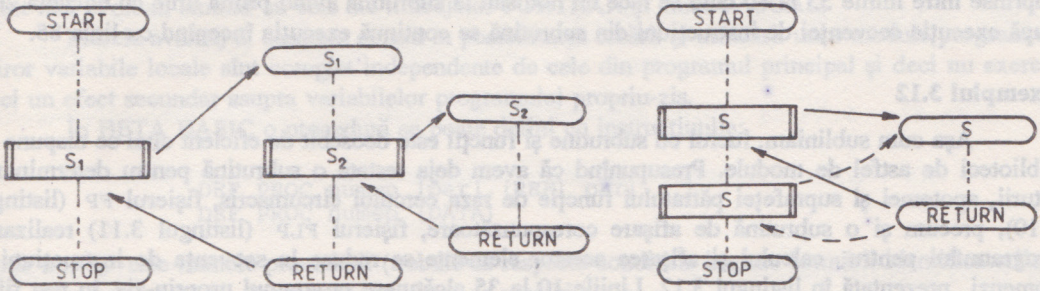


Fig. 3.2

În figura 3.2 este prezentat, utilizînd schemele logice, modul de apel şi revenirea din subrutine în configuraţiile acceptate. Descriera în BASIC a apelurilor şi revenirilor din subrutine este ilustrată în figura 3.3.

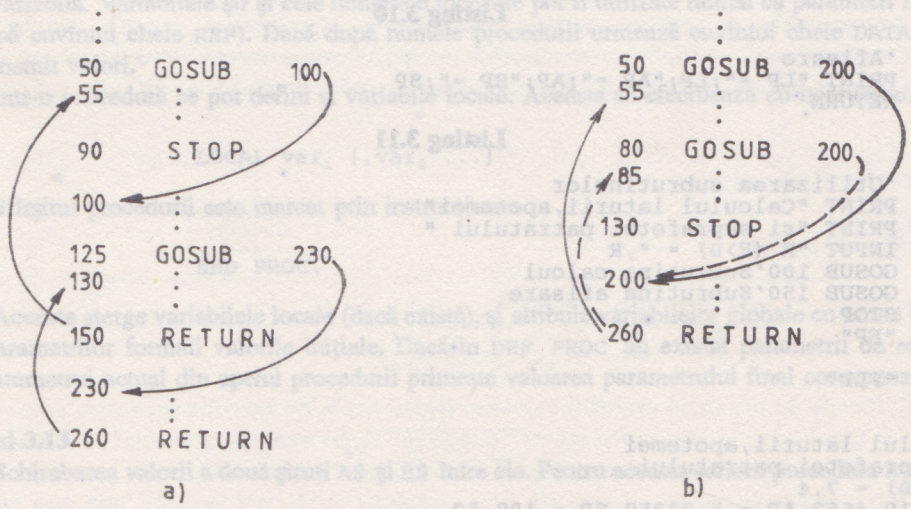


Fig. 3.3

În cazul secvenţelor prezentate în figura 3.2a execuţia decurge astfel: se execută instrucţiunile anterioare liniei 50; la întîlnirea instrucţiunii `GOSUB 100` din linia 50 se sare la linia 100 execuţia continuînd de aici pînă la linia 125, cînd se face un nou salt, de data aceasta la subrutina a cărei

primă linie are eticheta 230; după parcurgerea liniilor 230 la 260, la întîlnirea instrucţiunii RETURN (linia 260), se revine la instrucţiunea aflată imediat după ultimul salt la subrutina (efectuat la linia 125), respectiv la linia 130; se parcurg liniile 130 la 150, după care se revine, de data aceasta, la linia 55.

În secvenţa prezentată în figura 3.2b se execută instrucţiunile pînă la linia 50, se sare la linia 200 (salt la subrutină) se parcurge subrutina şi se revine la linia 55; se execută apoi instrucţiunile cuprinse între liniile 55 la 80 cînd se face un nou salt la subrutina avînd prima linie cu eticheta 200; după execuţia secvenţei de instrucţiuni din subrutină se continuă execuţia începînd cu linia 85.

Exemplul 3.12

Aşa cum subliniam, lucrul cu subrutine şi funcţii este deosebit de eficient cînd se dispune de biblioteci de astfel de module. Presupunînd că avem deja testată o subrutină pentru determinarea laturii, apotemei şi suprafeţei pătratului funcţie de raza cercului circumscris, fişierul FP (listingul 3.10), precum şi o subrutină de afişare corespunzătoare, fişierul FLP (listingul 3.11) realizarea programului pentru calculul şi afişarea acestor elemente se reduce la secvenţa de instrucţiuni şi comenzi prezentată în listingul 3.12. Liniile 10 la 35 alcătuiesc programul propriu-zis, în rest fiind conversaţie cu calculatorul.

```
100    'Calcul LP,AP,SP
105    LP=R*SQR(2)
110    AP=R*SQR(2)/2
115    SP=2*R^2
120    RETURN
```

Listing 3.10

```
150    'Afisare
155    PRINT "LP =";LP;"AP =";AP;"SP =";SP
160    RETURN
```

Listing 3.11

```
10     'Utilizarea subrutinelor
15     PRINT "Calculul laturii,apotemei"
16     PRINT "si suprafetei patratului "
20     INPUT "R (R>0) = ",R
25     GOSUB 100'Subrutina calcul
30     GOSUB 150'Subrutina afisare
35     STOP
MERGE "FP"
Ok
MERGE "FLP"
Ok
RUN
Calculul laturii,apotemei
si suprafetei patratului
R (R>0) = 7.4
LP = 10.4652 AP = 5.23259 SP = 109.52
Break in 35
Ok
```

Listing 3.12

Remarcă. *În majoritatea variantelor BASIC, lucrul cu subrutine are un mare neajuns: variabilele utilizate în interiorul acestora, aşa numiţii parametrii formali, sînt de fapt globale, adică,*

deşi sînt definite în subrutină, ele se regăsesc şi în programul apelant, ceea ce creează dificultăţi în apelul repetat, cu diferiţi parametrii actuali, ai aceleeaşi subrutine. În capitolele 4 şi 5 sînt prezentate exemple privitoare la modul de înlăturare, pe cît posibil, a acestui inconvenient.

● Subrutine BETA BASIC

Subliniem totuşi că varianta BETA BASIC permite lucrul cu subrutine (proceduri) la un nivel mult mai evoluat, similar cu cele ale limbajelor FORTRAN şi chiar PASCAL.

Marele avantaj al acestora constă în posibilitatea creării şi utilizării unor module program ale căror variabile locale sînt complet independente de cele din programul principal şi deci nu exercită nici un efect secundar asupra variabilelor programului propriu-zis.

În BETA BASIC o procedură se poate defini cu instrucţiunile:

```
DEF PROC numep [par[,[REF] par]...]
DEF PROC numep [DATA]
```

unde `numep` este numele procedurii (trebuie să respecte condiţiile impuse formării identificatorilor) `par` parametrii formali (trebuie să fie nume de variabile).

În cazul cînd numele parametrilor formali coincide cu nume de variabile din programul apelant atunci, pe perioada execuţiei procedurii, aceştia sînt salvaţi iar după revenirea din procedură, restauraţi astfel încît valorile lor nu sînt afectate.

Dacă înaintea unui parametru formal apare însă cuvîntul `REF`, atunci, la sfîrşitul procedurii valoarea parametrului formal respectiv este atribuită parametrului actual corespunzător care trebuie să fie o variabilă. Variabilele şir şi cele numerice indexate pot fi utilizate numai ca parametri referinţă (deci după cuvîntul cheie `REF`). Dacă după numele procedurii urmează cuvîntul cheie `DATA` atunci nu se transmit valori.

Într-o procedură se pot defini şi variabile locale. Aceasta se efectuează cu instrucţiunea

```
LOCAL var1 [,var2 ...]
```

Sfîrşitul procedurii este marcat prin instrucţiunea

```
END PROC.
```

Aceasta şterge variabilele locale (dacă există), şi atribuie variabilelor globale cu nume identice cu ale parametrilor formali valorile iniţiale. Dacă în `DEF PROC` au existat parametrii de referinţă, atunci parametrul actual din apelul procedurii primeşte valoarea parametrului final corespunzător.

Exemplul 3.13

Schimbarea valorii a două şiruri `A$` şi `B$` între ele. Pentru aceasta scriem procedura `SCHIMBA`:

```
1000 DEF PROC SCHIMBA REF A$,REF B$
1010   LOCAL AUX$
1020   AUX$=A$
1030   A$=B$
1040   B$=AUX$
1050 END PROC
```

Am folosit variabila auxiliară AUX$ pe care am definit-o ca locală şi variabilele A$ şi B$ care constituie parametri formali ai procedurii.

Apelul procedurii într-un program se efectuează prin enunţarea numelui ei urmat de cele ale parametrilor actuali.

Dacă înaintea execuţiei secvenţei:

```
10   DATA "SIRA","SIRB","SIRC"
20   READ A$,B$,C$
30   SCHIMBA A$,B$
40   SCHIMBA B$,C$
```

A$ = SIRA, B$ = SIRB, C$ = SIRC, după execuţia sa variabilele A$, B$ şi C$ vor conţine valorile:

A$ = SIRB, B$ = SIRC respectiv C$ = SIRA.

3.8 Instrucţiuni de control

Limbajul BASIC cuprinde un set de instrucţiuni de control care acoperă o bună parte din structurile de control cu care operează programarea structurată, precum şi instrucţiuni de control nestructurate.

În categoria instrucţiunilor de control structurate intră pe lîngă instrucţiunile pentru structuri liniare, cele destinate descrierii structurilor alternative şi repetitive.

Instrucţiunile de control nestructurate cuprind în principal salturile condiţionate şi necondiţionate. Totuşi, utilizate cu atenţie, acestea pot servi la crearea structurilor prezentate în capitolul 2.3, cu atît mai mult cu cît, din acest punct de vedere limbajul BASIC este deficitar.

● Instrucţiuni alternative

Sînt de formele:

```
IF expl THEN ina [ELSE inf]
IF expl THEN nrla [ELSE nrlf]
IF expl GOTO nrla [ELSE nrlf]
```

şi permit, funcţie de rezultatul evaluării expresiei logice expl, fie execuţia secvenţei ina sau salt la linia nrla (dacă expl are valoarea logică 'adevarat'), fie execuţia secvenţei inf, respectiv salt la linia nrlf (dacă expl are valoarea logică 'fals'), cînd ramura ELSE este prezentă.

Într-o instrucţiune IF, secvenţele ina, respectiv inf pot conţine una sau mai multe instrucţiuni separate prin caracterul ':', fără a depăşi însă lungimea maximă a unei linii.

Deoarece, în majoritatea variantelor BASIC, nu există un cuvînt cheie care să semnaleze sfîrşitul unei instrucţiuni IF, ci doar caracterul <CR>, care semnalează şi sfîrşitul liniei, pe aceeaşi linie BASIC, nu pot fi scrise după aceasta alte instrucţini. În cazul cînd acestea există, ele sînt considerate incluse în IF.

De exemplu, în secvenţa

```
60   IF A = 0 THEN B=B+1 ELSE B=0 : C=C+2 <CR>
```

dacă expresia `A=0` este adevărată se execută atribuirea `B=B+1` şi se trece la linia următoare, în caz contrar, se execută instrucţiunile `B=0` şi `C=C+2`.

Observaţie. Caracterul ':' utilizat aici are rolul de a separa cele două instrucţiuni conţinute în ramura `ELSE` şi nu de a delimita instrucţiunea

`IF A=0 THEN B=B+1 ELSE B=0` ***de instrucţiunea*** *`C=C+2`.*

În BASIC instrucţiunile alternative trebuie deci considerate astfel:

```
IF expl THEN instr ELSE instr <CR>.
```

Exemplul 3.14

Să scriem un program care să determine maximul din trei valori numerice.

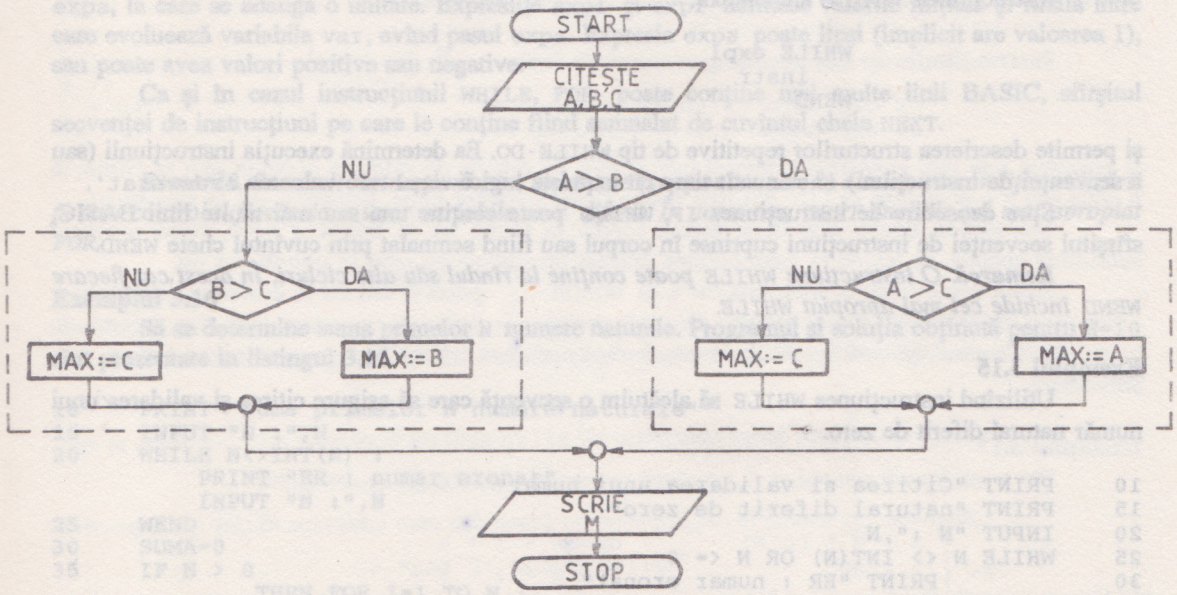


Fig. 3.4

Schema logică este cea din figura 3.4, iar programul şi soluţiile obţinute sînt prezentate în listingul 3.13.

```
10    PRINT "Maxim din 3 valori"
15    INPUT "A :",A
20    INPUT "B :",B
25    INPUT "C :",C
30    IF A>B
        THEN IF A>C
               THEN MAX=A
                    ELSE MAX=C
             ELSE IF B>C
                    THEN MAX=B
                    ELSE MAX=C
35    PRINT "MAX =";MAX
40    END
```

```
RUN
Maxim din 3 valori
A :3
B :5
C :6
MAX = 6
Ok
```

Listing 3.13

● Instrucţiuni repetitive

Din această categorie limbajul BASIC dispune de instrucţiunile `WHILE` şi `FOR`.

Instrucţiunea WHILE are sintaxa:

```
WHILE expl
   instr
WEND
```

şi permite descrierea structurilor repetitive de tip `WHILE-DO`. Ea determină execuţia instrucţiunii (sau a secvenţei de instrucţiuni) `instr` atît timp cît expresia logică `expl` are valoarea `'adevarat'`.

Spre deosebire de instrucţiunea `IF`, `WHILE` poate conţine una sau mai multe linii BASIC, sfîrşitul secvenţei de instrucţiuni cuprinse în corpul sau fiind semnalat prin cuvîntul cheie `WEND`.

Remarcă. *O instrucţiune `WHILE` poate conţine la rîndul său alte cicluri. În acest caz fiecare `WEND` închide cel mai apropiat `WHILE`.*

Exemplul 3.15

Utilizînd instrucţiunea `WHILE` să alcătuim o secvenţă care să asigure citirea şi validarea unui număr natural diferit de zero.

```
10    PRINT "Citirea si validarea unui numar"
15    PRINT "natural diferit de zero"
20    INPUT "N :",N
25    WHILE N <> INT(N) OR N <= 0
30          PRINT "ER : numar eronat"
35          INPUT "N :",N
40    WEND
45    PRINT "N =";N;" valoare corecta"
50    END
RUN
Citirea si validarea unui numar
natural diferit de zero
N :-1
ER : numar eronat
N :2.3
```

```
ER : numar eronat
N :5
N = 5  valoare corecta
Ok
```

Listing 3.14

Secvenţa de program şi testarea ei sînt prezentate în listingul 3.14. Instrucţiunea `WHILE` cuprinde liniile 25 la 40, conţinînd în corpul său alte două instrucţiuni: `PRINT` (linia 30), respectiv `INPUT` (linia 35). Aceasta din urmă este necesară pentru a modifica, dacă se doreşte, valoarea expresiei logice din linia 25; altfel o dată intrat în `WHILE`, programul ciclează. Dacă `N` este corect (este întreg şi mai mare decît zero) liniile 25 la 40 nu sînt executate niciodată.

Instrucţiunea `FOR` are sintaxa:

```
FOR var = expi TO expf [STEP exps]
  instr
NEXT [var]
```

şi permite descrierea în BASIC a structurilor repetitive cu număr cunoscut de paşi (vezi capitolul 2.3). Numărul de ori de cît se execută instrucţiunea (secvenţa de instrucţiuni) `instr` este dat de cîtul împărţirii întregi a diferenţei dintre valorile expresiilor numerice `expf` şi `expi`, la aceea a expresiei `exps`, la care se adaugă o unitate. Expresiile `expi` şi `expf` definesc valorile iniţială şi finală între care evoluează variabila `var`, avînd pasul `exps`. Expresia `exps` poate lipsi (implicit are valoarea 1), sau poate avea valori pozitive sau negative.

Ca şi în cazul instrucţiunii `WHILE`, `FOR` poate conţine mai multe linii BASIC, sfîrşitul secvenţei de instrucţiuni pe care le conţine fiind semnalat de cuvîntul cheie `NEXT`.

Remarcă. *Se admit instrucţiuni* `FOR` *imbricate (conţinute unele în altele), cu condiţia utilizării pentru controlul fiecăreia a unor variabile* `var` *diferit. În acest caz,* `NEXT` *închide cel mai apropiat* `FOR`.

Exemplul 3.16

Să se determine suma primelor `N` numere naturale. Programul şi soluţia obţinută pentru `N=10` sînt prezentate în listingul 3.15.

```
10    PRINT "Suma primelor N numere naturale"
15    INPUT "N :",N
20    WHILE N<>INT(N) :
          PRINT "ER : numar eronat" :
          INPUT "N :",N
25    WEND
30    SUMA=0
35    IF N > 0
            THEN FOR I=1 TO N :
                     SUMA=SUMA+I :
                 NEXT
40    PRINT "Suma =";SUMA
45    END
RUN
Suma primelor N numere naturale
N :10
Suma = 55
Ok
```

Listing 3.15

Suma se efectuează (linia 35) numai dacă `N>0`. Valoarea acestuia este validată de instrucţiunea `WHILE` din linia 20.

● Instrucţiuni de salt

Pot fi de salt **necondiţionat** sau **condiţionat**.

Instrucţiunea de salt necondiţionat este:

```
GOTO nrlin
```

şi determină saltul la linia cu numărul `nrlin`. La întîlnirea acestei instrucţiuni execuţia programului continuă cu linia `nrlin`, dacă este o instrucţiune executabilă, sau cu prima instrucţiune executabilă de după linia specificată.

Remarcă. Deşi, de multe ori, dintr-o aparentă simplitate, există tendinţa de a folosi această instrucţiune în controlul programelor, totuşi, pentru realizarea unor programe uşor de urmărit, de depanat şi, de ce nu, elegante, se recomandă evitarea utilizării ei, prin descrierea cu ajutorul instrucţiunilor structurate a eventualelor ramificaţii ale programului.

Instrucţiunile de salt necondiţionat sînt de forma:

```
ON exp GOTO nrlin1 [,nrlin2 ...]
ON exp GOSUB nrlin1 [,nrlin2 ...]
```

şi presupun mai întîi evaluarea expresiei întregi `exp` şi, funcţie de valoarea acesteia, salt la prima, la a doua sau a n-a valoare `nrlin` din listă (semnificînd numărul liniei, respectiv salt la subrutină).

Deosebirea dintre cele două forme constă în faptul că în forma a doua, după execuţia subrutinei apelate, se revine la instrucţiunea imediat următoare, pe cînd în primul caz nu.

Saltul condiţionat la subrutina `ON exp GOSUB ...` poate fi utilizat la descrierea structurilor alternative generalizate (capitolul 2.3). Datorită revenirii la linia imediat următoare, determinată de lucrul cu subrutine, această instrucţiune are de fapt o singură intrare şi o singură ieşire, şi deci, este structurată.

Exemplul 3.17

Fie `N` un număr natural şi `A, B` două numere reale date. Să se determine:

$$F=\begin{cases} N + B * Ln(N) & \text{dacă } N = 3 * k \\ A * \sqrt{N} & \text{dacă } N = 3 * k + 1 \\ A / B * N & \text{dacă } N = 3 * k + 2 \text{ cu } k \end{cases}$$

Pentru aceasta vom utiliza instrucţiunea `ON exp GOSUB ...`, unde `exp` este `N MOD 3 + 1`, luînd deci valori în mulţimea {1,2,3}. Programul sursă şi soluţiile obţinute sînt prezentate în listingul 3.16.

```
10    PRINT "Calculul valorii functiei F"
15    DATA 1.5,.5
20    READ A,B
```

```
30      INPUT "N :",N
35      WHILE N<>INT(N) OR N<0 :
                  PRINT "ER : numar eronat" :
                  INPUT "N =",N :
            WEND
40      ON N MOD 3 + 1 GOSUB 60,70,80
45      PRINT "F =";F
50      END
60      'Subrutina 1
65      F=N+B*LOG(N) : RETURN
70      'Subrutina 2
75      F=A*SQR(N) : RETURN
80      'Subrutina 3
85      F=A/B*N : RETURN
RUN
Calculul valorii functiei F
N :6
F = 6.89588
Ok
RUN
Calculul valorii functiei F
N :7
F = 3.96863
Ok
RUN
Calculul valorii functiei F
N :8
F = 24
Ok
```

Listing 3.16

O instrucţiune utilă, de salt condiţionat, este şi:

```
ON ERROR GOTO nrlin
```

care determină, în caz de eroare, salt la o rutină de tratare a erorilor scrisă de utilizator.

În diferite variante de BASIC întîlnim şi alte instrucţiuni de salt condiţionat, cum sînt:

```
ON KEY nr GOSUB nrlin(*)
```

salt la subrutina care debutează cu linia `nrlin` la acţionarea tastei funcţionale `nr`;

```
ON TIMER nr GOSUB nrlin(*)
```

întrerupere şi salt la subrutina care debutează cu linia `nrlin` la fiecare `nr` secunde;

```
ON PLAY nr GOSUB nrlin(*)
```

salt la subrutina specificată prin `nrlin` de tratare a evenimentului provocat de diminuarea numărului de note din zona tampon de melodii sub `nr` note;

```
ON COM nr GOSUB nrlin(*)
```

salt la subrutina specificată prin `nrlin` de tratare a evenimentului provocat de sosirea caracterelor în zona tampon de comunicaţii a canalului `nr`;

```
ON BREAK CONT(**)
```

anulează acţiunea tastei <ESC>, împiedicînd oprirea execuţiei programului;

```
ON BREAK GOSUB nrlin(**)
```

salt la subrutina specificată prin nrlin, la apăsarea de două ori a tastei <ESC>;

```
ON BREAK STOP(**)
```

determină oprirea programului la acţionarea tastei <ESC>.

Pentru descriera cît mai sugestivă şi mai firească a unui algoritm utilizînd limbajul BASIC, instrucţiunile de control pot fi combinate între ele (în sensul că pot fi conţinute unele în altele), în cadrul uneia sau mai multor linii BASIC. O astfel de utilizare trebuie efectuată însă cu multă atenţie şi cu respectarea riguroasă a modului de constituire şi execuţie a instrucţiunilor. În caz contrar, deşi aparent corect, programul poate furniza soluţii eronate.

Exemplul 3.18

Să se alcătuiască un program pentru determinarea valorii maxime şi minime dintr-un tablou unidimensional A() de 6 elemente. O posibilă soluţionare a problemei este ilustrată în schema logică din figura 3.5.

Întocmim după aceasta programul BASIC prezentat în listingul 3.17. Lansat în execuţie acesta nu dă întotdeauna soluţii corecte; astfel dacă variabila indexată A() conţine şirul de valori 1, 0, 3, 4, 5, 6, soluţia este eronată. Aceasta se datorează faptului că, dacă pentru I=1, în urma evaluării expresiei A(I)>MAX (0>1) rezultă, pentru aceasta, valoarea logică 'fals', efectuîndu-se instrucţiunile de pe ramura ELSE (unde cum 0 < 1, => MIN=0), inclusiv NEXT, pentru I=2, expresia A(2)>MAX (3>1), fiind adevărată, se efectuează atribuirea MAX=3 şi, întîlnindu-se cuvîntul ELSE, se iese din linia BASIC 50 (sărindu-se de fapt peste NEXT).

```
10    'Un program gresit
15    DIM A(5)
20    FOR I=0 TO 5
25    PRINT "A(";I;")";
30    INPUT " =",A(I)
35    NEXT
40    MAX=A(0)
45    MIN=A(0)
50    FOR I=1 TO 5 :
          IF A(I)>MAX
            THEN MAX=A(I)
            ELSE IF A(I)<MIN
                   THEN MIN=A(I) :
      NEXT
55    PRINT "MAX =";MAX;"MIN =";MIN
60    END
```

```
RUN
A( 0 ) =1
A( 1 ) =0
A( 2 ) =3
A( 3 ) =4
A( 4 ) =5
A( 5 ) =6
MAX = 3 MIN = 0
Ok
```

Listing 3.17

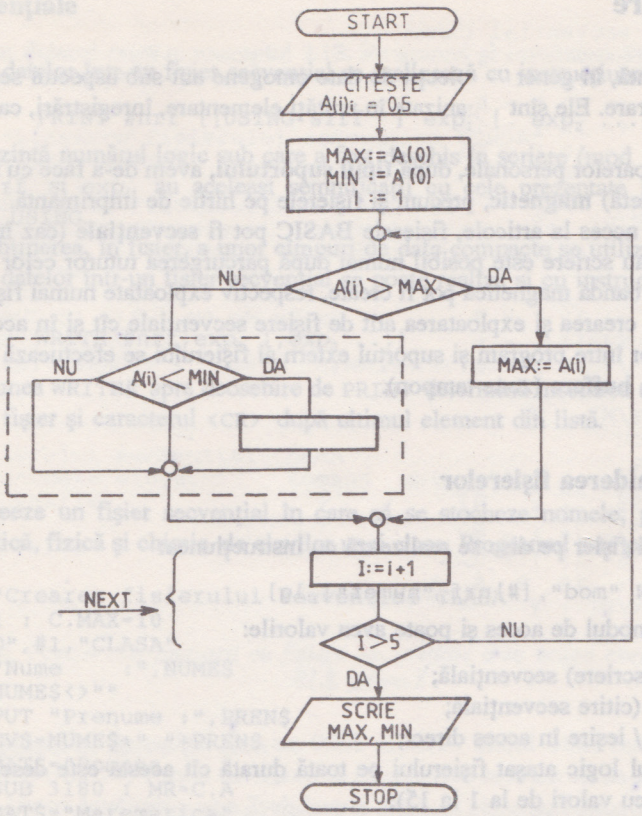


Fig. 3.5

Remediere. *Se scrie instrucţiunea* NEXT *pe o linie BASIC separată. Programul sursă corect şi execuţia acestuia sînt prezentate în listingul 3.18.*

```
10      'Programul corect
15      DIM A(5)
20      FOR I=0 TO 5
25      PRINT "A(";I;")";
30      INPUT " =",A(I)
35      NEXT
40      MAX=A(0)
45      MIN=A(0)
50      FOR I=1 TO 5 :
            IF A(I)>MAX
              THEN MAX=A(I)
              ELSE IF A(I)<MIN
                THEN MIN=A(I)
51      NEXT
55      PRINT "MAX =";MAX;"MIN =";MIN
60      END
RUN
A( 0 ) =1
A( 1 ) =0
A( 2 ) =3
A( 3 ) =4
A( 4 ) =5
A( 5 ) =6
MAX = 6 MIN = 0
Ok
```

Listing 3.18

3.9 Lucrul cu fişiere

Fişierele reprezintă, în gener olecţii de date omogene atît sub aspectul semnificaţiei cît şi al necesităţilor de prelucrare. Ele sînt anizate în unităţi elementare, înregistrări, care, la nivel logic se numesc articole.

În cazul calculatoarelor personale, după **tipul suportului**, avem de-a face cu fişiere pe **bandă** (casetă) şi pe **disc** (dischetă) **magnetic**, precum şi fişierele pe hîrtie de imprimantă.

După **modul de acces** la articole, fişierele BASIC pot fi **secvenţiale** (caz în care accesul la un articol pentru citire sau scriere este posibil numai după parcurgerea tuturor celor anterioare) şi în **acces direct**. Pe / de pe, bandă magnetică pot fi create, respectiv exploatate numai fişiere secvenţiale. Discul magnetic permite crearea şi exploatarea atît de fişiere secvenţiale cît şi în acces direct.

Transferul datelor între program şi suportul extern al fişierului se efectuează cu ajutorul unor zone de memorie numite **buffere** (zone tampon).

● Deschiderea şi închiderea fişierelor

Deschiderea unui fişier pe disc se realizează cu instrucţiunea:

```
OPEN "mod",[#]nrf,"numef"[,lg]
```

unde: `"mod"` specifică modul de acces şi poate avea valorile:

- `O` - ieşire (scriere) secvenţială;
- `I` - intrare (citire secvenţială;
- `R` - intrare / ieşire în acces direct;
- `nrf` - numărul logic ataşat fişierului pe toată durata cît acesta este deschis (expresie întreagă cu valori de la 1 la 15);
- `numef` - numele fişierului;
- `lg` - specifică lungimea în octeţi a înregistrării în cazul fişierelor cu acces direct.

Observaţie. *Se admite deschidera simultană a aceluiaşi fişier sub mai multe numere* `nrf`*, pentru citirea secvenţială sau în acces direct, dar numai cu n singur număr pentru scriere secvenţială.*

Închidera fişierelor pe disc se realizează cu instrucţiunea:

```
CLOSE [#]nrf1 [,[#]nrf2 ...]
```

unde: nrf_1 numerele logice sub care au fost deschise fişierele respective.

După închiderea unui fişier `nrf` poate fi utilizat la deschiderea aceluiaşi sau a altui fişier.

Instrucţiunea `CLOSE` fără argumente (parametri) determină închiderea tuturor fişierelor deschise.

Observaţie. *Instrucţiunea* `END` *şi comanda* `NEW` *închid automat toate fişierele deschise. Instrucţiunea* `STOP` *nu închide fişiere.*

● Fişiere secvenţiale

Scrierea datelor într-un fişier secvenţial se realizează cu instrucţiunea:

PRINT #nrf [[USING sirf] exp_1 [exp_2 ...]]

unde: nrf reprezintă numărul logic sub care a fost deschis în scriere (mod "O") fişierul.

Şirul sirf şi exp_1 au aceleaşi semnificaţii cu cele prezentate la instrucţiunile PRINT, respectiv PRINT USING.

Pentru obţinerea, în fişier, a unor cîmpuri de date compacte se utilizează caracterul ';'.

Scrierea datelor într-un fişier secvenţial se poate realiza şi cu instrucţiunea:

WRITE #nrf,exp_1 [,exp_2 ...]

Instrucţiunea WRITE# spre deosebire de PRINT determină inserarea unei virgule după fiecare valoare scrisă în fişier şi caracterul <CR> după ultimul element din listă.

Exemplul 3.19

Să se creeze un fişier secvenţial în care să se stocheze numele, prenumele şi mediile la română, matematică, fizică şi chimie ale elevilor unei clase. Programul este prezentat în listingul 3.19.

```
 5    PRINT "Crearea fisierului secvential CLASA"
10    C.MIN=1 : C.MAX=10
15    OPEN "O",#1,"CLASA"
20    INPUT "Nume     :",NUME$
25    WHILE NUME$<>""
30        INPUT "Prenume :",PREN$
35        ELEV$=NUME$+" "+PREN$
40        C.MAT$="Romana     "
45        GOSUB 3180 : MR=C.A
50        C.MAT$="Matematica"
55        GOSUB 3180 : MM=C.A
60        C.MAT$="Fizica     "
65        GOSUB 3180 : MF=C.A
70        C.MAT$="Chimie     "
75        GOSUB 3180 : MC=C.A
80        PRINT ELEV$;" : ";MR;MM;MF;MC
85        INPUT "Articol corect ? (Y/N) : ",C$
90        IF C$ = "Y" OR C$ = "y"
            THEN WRITE #1,ELEV$,MR,MM,MF,MC
95        INPUT "Nume     :",NUME$
100   WEND
105   END
3180  'Citeste si valideaza un numar intreg
3181  'cuprins intre C.MIN si C.MAX
3185  PRINT C.MAT$;"(";C.MIN;"<= N <=";C.MAX;")";
3190  INPUT " = ",C.A
3195  WHILE C.A<C.MIN OR C.A>C.MAX OR C.A<>INT(C.A):
          PRINT "ER : valoare eronata" :
          PRINT C.MAT$;"(";C.MIN;"<= N <=";C.MAX;")";:
          INPUT " = ",C.A :
      WEND
3200  RETURN
```

```
RUN
Crearea fisierului secvential CLASA
Numele     :Alexandrescu
Prenumele  :Vasile
Romana     ( 1 <= N <= 10 ) = 9
Matematica ( 1 <= N <= 10 ) = 10
Fizica     ( 1 <= N <= 10 ) = 10
Chimie     ( 1 <= N <= 10 ) = 9
Alexandrescu Vasile :  9  10  10  9
Articol corect ? (Y/N) : Y
Numele     :Florea
Prenumele  :Doina
Romana     ( 1 <= N <= 10 ) = 9
Matematica ( 1 <= N <= 10 ) = 8
Fizica     ( 1 <= N <= 10 ) = 9
Chimie     ( 1 <= N <= 10 ) = 7
Florea Doina :  9  8  9  7
Articol corect ? (Y/N) : Y
Numele     :Radulescu
Prenumele  :Luminita
Romana     ( 1 <= N <= 10 ) = 10
Matematica ( 1 <= N <= 10 ) = 9
Fizica     ( 1 <= N <= 10 ) = 9
Chimie     ( 1 <= N <= 10 ) = 9
Radulescu Luminita :  10  9  9  9
Articol corect ? (Y/N) : Y
Numele     :
Ok
```

Listing 3.19

Subrutina cuprinsă între liniile 3180 la 3200 asigură citirea şi validarea mediilor (valori rotunjite); acestea trebuie să fie numere întregi şi să se încadreze între 1 (`C.MIN`) şi 10 (`C.MAX`), valori precizate în linia 10. Articolul este scris în fişier numai dacă este considerat corect (liniile 85 şi 90).

Observaţie. *Deoarece închiderea fişierului se efectuează la întîlnirea instrucţiunii* `END`*, nu mai este necesară, în acest caz, scrierea explicită a unui* `CLOSE`*.*

Citirea datelor dintr-un fişier secvenţial se realizează cu instrucţiunea:

```
INPUT #nrf,var1 [,var2 ...]
```

unde: `nrf` numărul logic sub care a fost dechis în citire fişierul (mod `"I"`); var_i lista variabilelor care primesc valori în ordinea specificată în listă.

Observaţie. *Între tipul şi ordinea acestor variabile şi reprezentarea lor în fişier trebuie să fie o corespondenţă biunivocă.*

Atribuirile se fac astfel: primul caracter din fişier diferit de `<CR>`, spaţiu ( ) sau `<LF>`, reprezintă începutul unui număr, sfîrşitul lui fiind considerat caracterul ce precede `<CR>`, spaţiu sau virgulă. Dacă se întîlneşte caracterul ghilimele `(")`, atunci urmează un şir de caractere încheiat de un alt caracter ghilimele sau de sfîrşitul fişierului.

Exemplul 3.20

Să listăm fişierul creat în exemplul 3.19. Programul şi rezultatele execuţiei sînt prezentate în listingul 3.20.

```
 5     PRINT "Listarea fisierului secvential CLASA"
10     OPEN "I",#1,"CLASA"
15     IF EOF(1)=-1
          THEN PRINT "Fisierul CLASA este vid"
          ELSE PRINT "Numele si prenumele elevului";:
             PRINT "    romana  matematica     fizica";:
             PRINT "    chimie"
20     WHILE EOF(1)<>-1 :
          INPUT #1,ELEV$,MR,MM,MF,MC :
          PRINT USING "\                          \";ELEV$;:
          PRINT USING "      ##      ";MR;MM;MF;MC :
       WEND
25     END
RUN
Listarea fisierului secvential CLASA
Numele si prenumele elevului     romana  matematica     fizica     chimie
Alexandrescu Vasile                   9          10         10          9
Florea Doina                          9           8          9          7
Radulescu Luminita                   10           9          9          9
Ok
```

Listing 3.20

Remarcă. *Pentru alte operaţii cu fişiere secvenţiale a se vedea exemplele din capitolul 5.6.*

● Lucrul cu fişiere în acces direct

După cum am văzut, deschiderea fişierelor în acces direct se realizează cu instrucţiunea `OPEN` cu `'mod=R'`, ceea ce dă permisiunea atît de scriere cît şi de citire a fişierului specificat în `OPEN ("numef")` de număr logic `nrf`.

Spre deosebire de modul de lucru cu fişiere secvenţiale, în cazul fişierelor cu acces direct, înainte, respectiv după scrierea, respectiv citirea efectivă a unui articol din fişier, trebuie asigurat prin program transferul datelor în şi din zona tampon (în / din buffer).

Alocarea de buffere pentru fiecare fişier cu acces direct deschis, se realizează cu instrucţiunea:

```
FIELD [#]nrf[,dim AS varbf1 [,varbf2 ...]]
```

unde: `nrf` este numărul logic ataşat fişierului la deschidere; `dim` numărul de octeţi necesari pentru variabile; $varbf_i$ variabile de tip şir, semnificînd bufferul (bufferii) alocaţi fişierului.

Atenţie. *Numărul total al octeţilor alocaţi printr-o instrucţiune* `FIELD` *nu trebuie să depăşească lungimea înregistrarii* `lg` *specificate la deschiderea fişierului respectiv.*

Observaţie. *Variabilele din lista instructiunii* `FIELD` *nu pot apare în instrucţiuni* `LET` *sau* `INPUT`.

Variabilele $varbf_i$ utilizate ca buffer constituie zona de memorie prin care se face schimbul de informaţii între program şi fişierul în acces direct.

Mutarea datelor din memorie într-un buffer de fişier în acces direct se realizează cu instrucţiunile:

```
LSET varbf1 [,varbf2 ...]=exp1 [,exp2 ...]
RSET varbf1 [,varbf2 ...]=exp1 [,exp2 ...]
```

unde: $varbf_i$ reprezintă numele variabilelor declarate cu FIELD; exp_i expresii cu şiruri de caractere (conţin deci cel mult operatorul '+', concatenare).

O variabilă numerică, pentru a fi depusă în buffer, trebuie convertită în caracter.

Dacă şirul rezultat în urma evaluării expresiilor exp_i necesită mai puţini octeţi decît au fost rezervaţi cu FIELD pentru lista de variabile varbf, atunci LSET îl aliniază la stînga iar RSET la dreapta, restul cîmpurilor fiind completate cu spaţii.

În afara instrucţiunilor LSET şi RSET efectuarea de atribuiri asupra variabilelor utilizate ca buffer se mai poate executa cu instrucţiunile PRINT#, PRINT# USING şi WRITE#. În cazul instrucţiunii WRITE#, zona buffer neutilizată este umplută cu spaţiu după <CR>.

Scrierea efectivă a unui articol într-un fişier în acces direct se realizează cu instrucţiunea:

```
PUT [#]nrf[,nring]
```

unde: nrf este numărul ataşat fişierului; nring dacă există, semnifică numărul înregistrării din fişier unde se va face scrierea; dacă lipseşte, atunci înregistrarea va primi numărul următor celui dat ultimei înregistrări cu care s-a operat în PUT; nring poate lua valori întregi de la 1 la 32767.

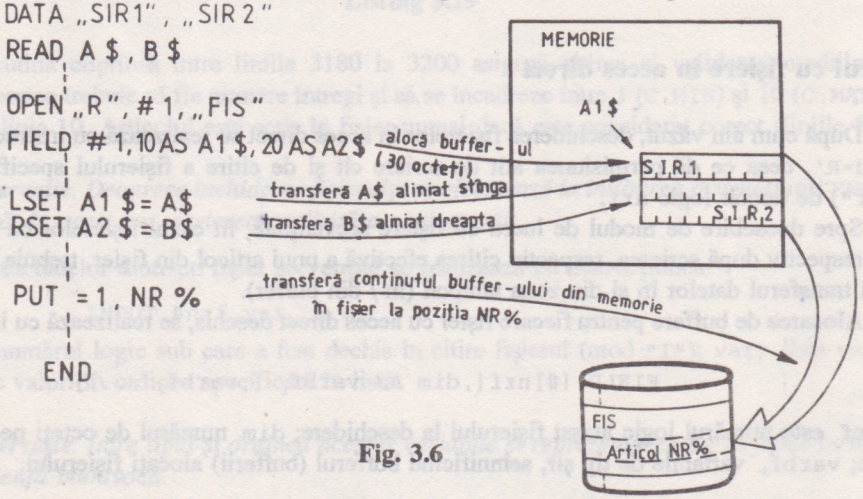


Fig. 3.6

În figura 3.6 este prezentat schematic modul de pregătire şi efectuare a scrierii unei înregistrării într-un fişier în acces direct.

Exemplul 3.21

Să se creeze un fişier cu acces direct avînd structura articolelor similară celei din exemplul 3.19.

Indicaţie. Variabilele numerice utilizate la memorarea mediilor (rotunjite), pentru a putea fi scrise în fişier trebuie convertite în variabile caracter. Utilizăm în acest sens funcţia `STR$(x)`.

Programul este prezentat în listingul 3.21.

```
 5     PRINT "Crearea fisierului cu acces direct CLASAD"
10     C.MIN=1 : C.MAX=10
15     OPEN "R",#1,"CLASAD",40
20     FIELD #1,28 AS EL$,3 AS MR$,3 AS MM$, 3 AS MF$,3 AS MC$
25     INPUT "Numar inregistrare :",NR%
30     WHILE NR%>0
35         INPUT "Nume     :",NUME$
40         INPUT "Prenume :",PREN$
45         ELEV$=NUME$+" "+PREN$
50         C.MAT$="Romana     "
55         GOSUB 3180 : MR=C.A
60         C.MAT$="Matematica"
65         GOSUB 3180 : MM=C.A
70         C.MAT$="Fizica     "
75         GOSUB 3180 : MF=C.A
80         C.MAT$="Chimie     "
85         GOSUB 3180 : MC=C.A
90         PRINT ELEV$;" : ";MR;MM;MF;MC
95         INPUT "Articol corect ? (Y/N) : ",C$
100        IF C$="Y" OR C$="y"
               THEN LSET ELS$=ELEV$ :
                    RSET MR$=STR$(MR) :
                    RSET MM$=STR$(MM) :
                    RSET MF$=STR$(MF) :
                    RSET MC$=STR$(MC) :
                    PUT #1,NR%
105        INPUT "Numar inregistrare :",NR%
110    WEND
115    END
MERGE "C7"
Ok
RUN
Crearea fisierului cu acces direct CLASAD
Numar inregistrare :1
Numele     :Alexandrescu
Prenumele :Vasile
Romana     ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica     ( 1 <= N <= 10 ) = 10
Chimie     ( 1 <= N <= 10 ) = 9
Alexandrescu Vasile :  9  10  10  9
Articol corect ? (Y/N) : Y
Numar inregistrare :2
Numele     :Florea
Prenumele :Doina
Romana     ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 8
Fizica     ( 1 <= N <= 10 ) = 9
Chimie     ( 1 <= N <= 10 ) = 7
Florea Doina :  9  8  9  7
Articol corect ? (Y/N) : Y
Numar inregistrare :3
```

```
Numele     :Radulescu
Prenumele  :Luminita
Romana     ( 1 <= N <= 10 ) = 10
Matematica ( 1 <= N <= 10 ) = 9
Fizica     ( 1 <= N <= 10 ) = 9
Chimie     ( 1 <= N <= 10 ) = 9
Radulescu Luminita :  10  9  9  9
Articol corect ? (Y/N) : Y
Numar inregistrare :
Ok
```

Listing 3.21

Pentru citirea unei înregistrări dintr-un fişier în acces direct se utilizează, pe lîngă instrucţiunea FIELD (necesară pentru alocarea bufferului), instrucţiunea GET cu sintaxa:

```
GET [#]nrf[,nring]
```

unde `nrf` şi `nring` au aceleaşi semnificaţii ca la utilizarea lor în instrucţiunea PUT.

După o instrucţiune GET, pentru citirea caracterelor din buffer şi atribuirea valorilor variabilelor corespunzătoare, se pot utiliza instrucţiunile `INPUT#` şi `LINE INPUT#`.

Exemplul 3.22

Citirea selectivă a articolelor fişierului `"CLASAD"` creat în exemplul anterior şi calculul mediei generale a fiecărui elev.

Programul BASIC şi rezultatele execuţiei sînt prezentate în listingul 3.22

```
10    PRINT "Citire fisier in acces direct"
15    OPEN "R",#1,"CLASAD",40
20    FIELD #1,28 AS EL$,3 AS MR$,3 AS MM$, 3 AS MF$,3 AS MC$
25    INPUT "Numar inregistrare :",NR%
30    WHILE NR%>0
35        GET #1,NR%
40        MED=(VAL(MR$)+VAL(MM$)+VAL(MF$)+VAL(MC$))/4
45        PRINT EL$;"  medii:";VAL(MR$);VAL(MM$);VAL(MF$);
50        PRINT VAL(MC$);"=> medie generala :";MED
55        INPUT "Numar inregistrare :",NR%
60    WEND
65    END
RUN
Citire fisier in acces direct
Numar inregistrare :1
Alexandrescu Vasile          medii: 9  10  10  9 => medie generala : 9.5
Numar inregistrare :3
Radulescu Luminita           medii: 10  9  9  9 => medie generala : 9.25
Numar inregistrare :2
Florea Doina                 medii: 9  8  9  7 => medie generala : 8.25
Numar inregistrare :
Ok
```

Listing 3.22

3.10 Înlănţuirea programelor

BASIC oferă posibilitatea ca pe parcursul execuţiei unui program, la un moment dat, să se lanseze în execuţie un alt program cu sau fără transferul valorilor tuturor variabilelor sau numai a unora din acestea.

Întreruperea programului curent şi predarea controlului altui program se realizează cu comanda CHAIN (prezentată în capitolul 3.2).

Comunicarea între programe se poate realiza în diferite moduri. În continuare ne referim numai transmiterea valorilor variabilelor între programe (apelant-apelat şi invers). Aceasta se poate efectua fie utilizînd opţiunea ALL a comenzii CHAIN (caz în care se transmit toate variabilele) sau utilizînd instructiunea COMMON.

Aceasta are sintaxa:

```
COMMON lista
```

unde lista reprezintă lista variabilelor numerice sau de tip şir care se transmit programului apelat. Ele sînt depuse automat într-o arie comună, fiind accesibile tuturor programelor BASIC înlănţuite prin CHAIN.

Remarcă. *Variabilele indexate sînt specificate prin identificator urmat de grupul '()'; declararea lor în programul apelat nu mai este necesară.*

Atenţie. *Utilizarea în programul apelant a instrucţiunilor de definire explicită a tipului (DEFINT, DEFSNG, DEFDBL sau DEFSTR) are efect local. Pentru asigurarea concordanţei tipurilor variabilelor comune celor două programe, în programul apelat este necesară reutilizarea acestora asupra eventualelor variabile implicate.*

Exemplul 3.23

Să examinăm un exemplu pur demonstrativ.

În listingul 3.23 se prezintă programul PA în care se definesc două variabile: variabila X, iniţializată de la tastatură, şi variabila indexată A().

În linia 40 se face înlănţuirea cu programul PA1 (listingul 3.24) din care, la linia 30, se lansează programul PA11 (listingul 3.25).

În programul PA11 se introduc de la tastatură valorile variabilei indexate B() după care, în linia 30 se determină întoarcerea în PA1 (la linia 35). După atribuirile din linia 40, în linia 45 se revine în PA la instrucţiunea cu numărul de linie 45.

O execuţie a acestui lanţ de programe, pornind de la programul PA, este prezentată în listingul 3.26.

```
10    PRINT "Start program PA"
15    DEFINT A
20    DIM A(2)
25    COMMON A(),X
30    INPUT "X :",X
35    PRINT "Se lanseaza PA1"
40    CHAIN "PA1"
```

```
45    PRINT "Revenire in PA"
50    PRINT A(0);"^";X;"+";A(1);"^";X-1;"=";A(0)^X+A(1)^(X-1)
55    END
```

Listing 3.23

```
 5    PRINT "Start program PA1"
10    DEFINT A
15    DIM B(2)
20    COMMON A(),B(),X
25    PRINT "Se lanseaza PA11"
30    CHAIN "PA11"
35    PRINT "Revenire in PA1"
40    FOR I=0 TO 2 : A(I)=B(I) :
NEXT
45    CHAIN "PA",45
```

Listing 3.24

```
LOAD "PA"
RUN
Start program PA
X :4
Se lanseaza PA1
Start program PA1
Se lanseaza PA11
```

```
10    PRINT "Start program PA11"
20    COMMON B(),X
25    FOR I=0 TO 1 :
          PRINT "B(";I;")"; :
          INPUT " = ",B(I) :
      NEXT
30    CHAIN "PA1",35
```

Listing 3.25

```
Start program PA11
B ( 0 ) = 4
B ( 1 ) = 5
Revenire in PA1
Revenire in PA
 4 ^ 4 + 5 ^ 3 = 381
Ok
```

Listing 3.26

3.11 Elemente de grafică

Grafica automată, prin amploarea pe care a căpătat-o în ultimul deceniu, a devenit un domeniu special al informaticii. Evoluţia ei, şi a mijloacelor tehnice (echipamentelor specializate) este mai mult decît spectaculoasă. Implicaţiile sînt multiple reliefîndu-se şi în diferitele versiuni şi generaţii ale limbajului BASIC.

Dintre toate categoriile de instrucţiuni BASIC cele grafice cunosc o foarte mare varietate, dar şi o puternică dependenţă de versiunile existente, astfel încît este greu de realizat (şi puţin eficient) o tratare globală a lor. Totuşi, fiind un domeniu foarte important al informaticii, o prezentare, chiar schematică, a lor este absolut necesară.

Înainte de a prezenta, spre exemplificare, unele facilităţi grafice disponibile în BASIC-80 (GBASIC) sub CP/M şi GW-BASIC sub MS-DOS, se impune precizarea cîtorva noţiuni utilizate curent în aplicaţii grafice.

● Fereastra, vizor, scalare, decupare

Pentru a înlesni înţelegerea acestor noţiuni considerăm următoarele (figura 3.7):

Fie D un dreptunghi de dimensiune m*n (n > 0 şi m > 0) spaţiul maxim de desen la un moment dat, spaţiu în care am realizat (sau intenţionăm să realizăm) un desen. Dacă în acest spaţiu definim submulţimea F a dreptunghiului [a,b]·[c,d] cu 0<a, b≤m şi 0<c, d≤n spunem că am definit o fereastră (window) în spaţiul D. Atît dimensiunile spaţiului D cît şi cele ale ferestrei F, sînt exprimate în coordonate care ţin de specificul aplicaţiei (de mărimile şi unităţile de măsură efectiv utilizate).

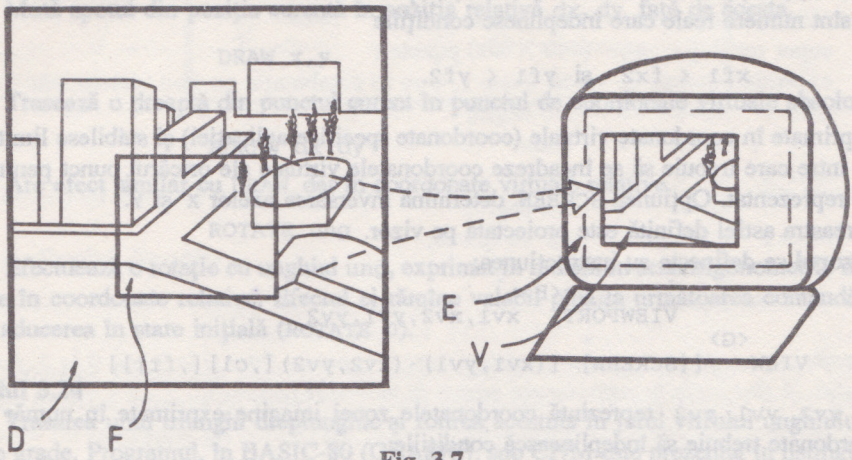


Fig. 3.7

Fie, în acelaşi timp, E suprafaţa totală de k*l puncte a spaţiului util de desen al echipamentului (de exemplu al displayului). O suprafaţă V dreptunghiulară [e,f]·[g,h] a acestuia (cu 0<a, b≤k şi 0<g, h≤l) se numeşte vizor (viewport).

***Observaţie.** Dimensiunile maxime ale spaţiului E (k şi l) depind de tipul dispozitivului grafic utilizat.*

Atît fereastra F cît şi vizorul V pot fi translatate oriunde în mulţimile D, respectiv E. De asemenea ele pot fi mărite sau micşorate după necesităţi, astfel încît pe ecran sau pe o parte a lui să avem la un moment dat întregul desen sau numai un detaliu al acestuia.

Reprezentările grafice presupun suprapunerea originii ferestrei F peste cea a vizorului V. Cum F şi V au dimensiuni diferite, rezultă că reprezentarea se va efectua la o anumită scară. Raporturile dintre dimensiunile laturilor ferestrei şi cele corespunzătoare ale vizorului se numesc **factori de scala**. Fie Sx şi Sy aceştia. În cazul în care avem de reprezentat suprafeţe atunci, dacă Sy <> Sx, imaginea conţinută în fereastră va fi reprezentată deformat pe vizor. Pentru evitarea deformărilor în acest caz, se va utiliza un singur factor de scalare:

```
S = max(Sx,Sy)
```

Încercarea de a desena în afara spaţiului ferestrei nu este semnalată ca eroare, dar părţile

desenului cere depăşesc limitele acesteia nu se reprezintă. Această operaţie se numeşte **decupare** (clipping).

Definirea ferestrei se efectuează cu instrucţiunea:

```
                    <B>
WINDOW    xf1,xf2,yf1,yf2
                    <G>
WINDOW    [[SCREEN] (xf1,yf1)-(xf2,yf2)]
```

unde `xf1, xf2, yf1, yf2` sînt limitele abscisei, respectiv ordonatei ferestrei. Coordonatele `xf1, xf2, yf1, yf2` sînt numere reale care îndeplinesc condiţiile

```
xf1 < fx2  şi yf1 < yf2.
```

Ele sînt exprimate în coordonate virtuale (coordonate specifice aplicaţiei) şi stabilesc limitele minime şi maxime între care trebuie să se încadreze coordonatele virtuale ale oricărui punct pentru ca acesta să poată fi reprezentat. Opţiunea `SCREEN` determină inversarea axelor `X` şi `Y`.

Fereastra astfel definită este proiectată pe vizor.

Vizorul se defineşte cu instrucţiunea:

```
                    <B>
VIEWPORT    xv1,xv2,yv1,yv2
<G>
VIEW    [[SCREEN] [(xv1,yv1)-(xv2,yv2)[,cl][,fr]]]
```

unde `xv1, xv2, yv1, yv2` reprezintă coordonatele zonei imagine exprimate în număr de puncte. Aceste coordonate trebuie să îndeplinească condiţiile:

```
0 < xv1, xv2 < k
0 < yv1, yv2 < l
```

unde `k` şi `l` reprezintă dimensiunile maxime în număr de puncte, ale spaţiului de desen al dispozitivului grafic respectiv; `cd` reprezintă codul culorii (0-3) cu care va fi umplut vizorul, iar `fr` solicită trasarea frontierei vizorului (cod culoare `fr`).

Dacă `SCREEN` este prezent atunci coordonatele grafice se vor calcula relativ la întregul ecran, în caz contrar sînt relative la vizor.

Atît fereastra cît şi vizorul pot fi redefinite pe parcursul unui program, însă, la un moment dat există o singură fereastră şi un singur vizor, definite prin ultimele instrucţiuni WINDOW şi VIEWPORT executate.

Limbajul GW-BASIC permite, de asemenea, definirea de ferestre de text cu instrucţiunea:

```
VIEW PRINT [nrl1  TO nrl2 ]
```

unde nrl_1 şi nrl_2 sînt expresii întregi cu valori între 1 şi 24 reprezentînd limitele acestuia în număr de linii. După definirea ferestrei de text toate funcţiile de scriere, poziţionare, editare, defilare au loc numai în acesta.

<B> BASIC-80; <G> GW-BASIC

● Instrucţiuni grafice

Principalele instrucţiuni grafice implementate în BASIC-80 (GBASIC) sînt:

```
MOVE x,y.
```

Mută spotul (punctul luminos de pe ecran) din poziţia în care se află în punctul de coordonate virtuale absolute (x,y).

```
RMOVE dx,dy
```

Mută spotul din poziţia curentă în poziţia relativă dx, dy faţă de acesta.

```
DRAW x,y
```

Trasează o dreaptă din punctul curent în punctul de coordonate virtuale absolute (x,y).

```
RDRAW dx,dy
```

Are efect similar cu DRAW dar în coordonate virtuale relative.

```
ROTATE ung
```

Efectuează o rotaţie cu unghiul ung, exprimat în radiani în sens trigonometric direct, a liniilor desenate în coordonate relative. Efectul ei rămîne valabil pînă la următoarea comandă ROTATE sau pînă la aducerea în stare iniţială (ROTATE 0).

Exemplul 3.24

Trasarea unui triunghi dreptunghic şi rotirea acestuia în jurul vîrfului unghiului drept din 30 în 30 de grade. Programul, în BASIC-80 (GBASIC), sub CP/M este prezentat în listingul 3.27. Liniile 15 la 35 permit citirea şi validarea dimensiunilor catetelor (subrutina 3180 a fost prezentată în exemplul 3.19). Variabila RGRD conţine valoarea în radiani a unui grad(180 =). Liniile 50 şi 100 determină comutarea ecranului din mod **defilare** în mod **pagina** şi invers.

```
 5    PRINT "Trasare si rotire triunghi dreptunghic"
10    C.MIN=10 : C.MAX=50 : RGRD=3.1415/180
15    C.MAT$="Cateta AB"
20    GOSUB 3180 : C=C.A
25    C.MAT$="Cateta AC"
30    GOSUB 3180 : B=C.A
35    PRINT CHR$(27)+"I";CHR$(27)+"2"
40    WINDOW 0,120,0,120
45    VIEWPORT 20,120,0,100
50    MOV 60,60
55    FOR I=0 TO 330 STEP 30
          ROTATE RGRD*I
          GOSUB 80
      NEXT
60    INPUT R$
65    PRINT CHR$(27)+"I";CHR$(27)+"2"
70    END
80    'Trasare triunghi dreptunghic
85    RDRAW C,0
90    RDRAW -C,B
95    RDRAW 0,-B
100   RETURN
```

Listing 3.27

Dintre principalele categorii de instrucţiuni grafice ale interpretorului GW-BASIC amintim:

A. Stabilire caracteristici ecran, cursor, culori

```
SCREEN [mod] [,ap] [,actp] [,vizp]
```

Stabileşte caracteristicile ecranului de afişat astfel:

`mod` expresie numerică cuprinsă între 0 şi 255 cu semnificaţiile:
`0` mod text (25 linii şi 80 coloane);
`1` rezoluţie medie <MR> (320 X 200 puncte);
`2` rezoluţie înaltă <HR> (640 X 200 puncte);
`>3` super rezoluţie <SR> (640 X 400 puncte);
`ap` expresie întreagă cu valori 0 sau 1, care în combinaţie cu `mod` determină:

`mod=0`	`ap=0`	=>	alb-negru;
	`ap=1`	=>	color;
`mod=1`	`ap=0`	=>	color
	`ap=1`	=>	alb-negru;
`mod>1`	=>		`ap` se ignoră;

`actp` selectare pagină activă;
`vizp` selectare pagină virtuală.

Observaţie. *`actp` şi `vizp` pot avea valori întregi cuprinse între 0 la 7 sau 0 la 3, funcţie de lăţimea ecranului, şi sînt luate în considerare numai în mod text (`mod=0`).*

```
LOCATE [rnd] [,[col] [,per] [,[lstart] [,lstop]]]     (în mod text)
LOCATE [rnd] [,[col] [,per] [,forma]]                 (în mod grafic)
```

Poziţionează cursorul în rîndul `rnd`, coloana `col` şi stabileşte caracteristicile de vizualizare ale cursorului `per` (0 - invizibil; 1 - permanent; 2 - 10 clipitor) şi dimensiunea acestuia în mod text `lstart` şi `lstop` (expresii numerice 0-13 pentru cursor utilizator şi de scriere forţată şi 32-45, modulo 32, pentru cursor utilizator în exclusivitate) sau forma sa `forma`, în mod grafic (se programează matricea cursorului).

```
COLOR [cl] [,fc]
COLOR<MR>   [fc] [,pl] [,cd] [,[fg] [,ct]]
COLOR<HR>    [cd] [,fg] [,ct]
COLOR<SR>    [cd] [,fg] [,ct]
```

Defineşte culoarea vizorului text `fc` (expresie numerică 0-31, modulo 15), paleta coloristică `pl` (expresie numerică 0-255, modulo 2), culoarea desenelor `cd` (expresie numerica 0-3), culoarea vizorului grafic `fg` (expresie numerică 0-3) şi culoarea caracterelor (a textului) `ct` (expresie numerică 0-1 în mod text sau 0-7 pentru `mod=1 <medie rezolutie>`).

Observaţie. *În mod înaltă rezoluţie (`mod=2` în `SCREEN` anterior), `ct` se pune în operaţie `XOR` (sau exclusiv) cu harta de alocare ecran. În mod superrezoluţie (`mod > 3`) `ct=0` semnifică scriere video normal, `ct=1` video invers, iar pentru `ct>1` se efectuează `XOR` cu harta de alocare ecran. În ambele cazuri parametrul `fg` poate lua numai valorile întregi 0 sau 1.*

Codurile culorilor sînt:

0 - negru	8 - gri
1 - albastru	9 - albastru aprins
2 - verde	10 - verde aprins
3 - cian	11 - cian aprins
4 - roşu	12 - roşu aprins
5 - magenta	13 - magenta aprins
6 - maron	14 - galben
7 - alb	15 - alb intens

```
CLS [n]
```

Umple cu culoarea fondului întregul ecran (`n=0`), vizorul grafic (`n=1`) sau vizorul text (`n=2`).

B. Trasări, haşurări

```
PSET [STEP] (x,y)[,cl]
PRESET [STEP] (x,y)[,cl]
```

Provoacă iluminarea punctului de coordonate absolute sau relative `(x,y)` în culoarea `cl` (expresie numerică întreagă cu valori 0-3). Dacă parametrul `cl` lipseşte atunci `PSET` afişează punctul cu culoarea `cd` declarată in `COLOR`, iar `PRESET` cu culoarea fondului (a vizorului grafic `<fg` in `COLOR>`) cea ce, de fapt, înseamnă ştergerea lui.

```
LINE [[STEP][(x1 ,y1 )]-[STEP](x2 ,y2 )][,cl][,B[F]][,stil]]
```

Trasează o linie între punctele de coordonate relative (dacă `STEP` este prezent) sau absolute (x_1, y_1) şi (x_2, y_2). Opţiunea `B` determină trasarea unui dreptunghi avînd colţurile de coordonate (x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2) cu culoarea `cl`, iar `BF` trasarea şi umplerea acestuia cu culoarea `cl`. `stil` este o valoare întreagă pe 16 biţi şi reprezintă şablonul de aprindere a punctelor (utilizat pentru linii întrerupte, punctate, etc.).

```
CIRCLE [STEP] (x,y),r[,cl[,start,stop[,ap]]]
```

Trasează un (arc de) cerc sau o elipsă de centru `(x,y)` şi rază `r` (calculate în pixeli), în coordonate absolute sau relative (opţiunea `STEP`), între unghiurile `start` şi `stop` (exprimate în radiani (`start, stop [-2 ,2 ]`), cu culoarea `cl` şi cu raport de aspect `asp` (implicit 5/6).

Observaţie. *Dacă* `start` *sau* `stop` *au valori negative se trasează şi raza.*

```
DRAW exps
```

Trasează o imagine grafică descrisă de comenzile GML aparţinînd expresiei şir `exps`.

Comenzile limbajului GML sînt:

a) stabilire factor de scară (`1/<k>`):

`S<k>` unde `<k>` este o expresie numerică întreagă cuprinsă între 1 şi 255 (implicit 4);

b) culori:

`C<cl>` stabileşte culoarea `cl` (0-3 pentru rezoluţie medie, 0-1 rezoluţie mare şi superrezoluţie);

`P<cl>,<cf>`
stabileşte culoarea interiorului `cl` şi a conturului (frontierei) `cf` ale unei figuri (`cl` şi `cf` respectă condiţiile comenzii `C`);

c) deplasări, trasări:

`B`	dacă precede comenzile următoare determină deplasarea fără trasare; dacă nu se efectuază şi trasare;
`U[<n>]`	deplasare în sus cu `<n>*<k>` puncte;
`D[<n>]`	deplasare în jos cu `<n>*<k>` puncte;
`L[<n>]`	deplasare la stînga cu `<n>*<k>` puncte;
`R[<n>]`	deplasare la dreapta cu `<n>*<k>` puncte;
`E[<n>]`	deplasare oblică (SV-NE) cu `<n>*<k>` puncte;
`M<x>,<y>`	deplasare spot în poziţia de coordonate `(x,y)` absolute sau relative (dacă `x` şi `y` sînt precedate de `'+'` sau `'-'`); `x` şi `y` sînt numere întregi;
`N`	deplasare în poziţia iniţială.

Remarcă. *GML oferă de asemenea posibilitatea lansării şi execuţiei unor comenzi specificate într-un subşir `subs` cu comanda: `X<subs>`.*

Exemplul 3.25

Trasarea în modul grafic medie rezoluţie a unui triunghi isoscel:

```
10   'Trasare triunghi isoscel
15   SCREEN 1
20   DRAW "BM30,30;E50;M90,30;M10,10"
```

Observaţie. *Caracterul `';'` utilizat în exemplul de mai sus pentru separarea comenzilor GML exte opţional.*

```
PAINT [STEP] (x,y)[,tapet[,contur[,fond]]]
```

Determină colorarea sau mozaicarea unei arii grafice începînd din punctul de coordonate absolute sau relative `(x,y)`, cu culoarea `tapet` (dacă este o expresie numerică întreagă cu valori 0-3) sau cu un model (dacă `tapet` este o expresie şir de 1-64 octeţi, formînd un şablon ce reprezintă măşti de 8 biţi). Coordonatele `x, y` trebuie să aparţină interiorului figurii de umplut. Conturul acestei figuri se va trasa cu culoarea `contur` (valoare numerică întreagă cu valori 0-3) sau, implicit, cu `tapet`.

C. Salvări şi restaurări de imagini

Salvarea imaginilor se realizează cu instrucţiunea:

```
GET [STEP] (x1, y1)-[STEP] (x2, y2),mat
```

Aceasta transferă imaginea grafică cuprinsă în dreptunghiul cu vîrfurile de coordonate (x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_1) într-un masiv `mat` de tip întreg care trebuie să aibă dimensiunea minimă (în octeţi):

```
4 + INT((dx * bpp +7)/8) * dy
```

unde `bpp` reprezintă numărul de biţi pe pixel (1 pentru rezoluţie mare şi super rezoluţie `<mod=2`, respectiv `mod=3` în `SCREEN>`, respectiv 2 pentru rezoluţie medie `<'mod'=1>`), iar `dx` şi `dy` reprezintă dimensiunile în puncte (pixeli) pe orizontală şi verticală ale zonei imagine de salvat.

Restaurarea imaginilor grafice salvate se realizează cu instrucţiunea:

```
PUT (x,y),mat[,verb]
```

Aceasta transferă pe ecran, într-un dreptunghi avînd coordonatele colţului de NV (stînga sus), perechea `(x,y)`, imaginea memorată în tabloul numeric `mat` interacţionîndu-se cu imaginea deja existentă funcţie de parametrul `verb`, ce poate lua valorile:

`PSET`	transferă punct cu punct, cu respectarea culorii din momentul salvării imaginii;
`PRESET`	transferă punct cu punct reversul imagini salvate;
`AND`	efectuează operaţia logică `SI` între elementul de imagine existent şi cel salvat în `mat`;
`OR`	suprapune imaginea salvată peste cea existentă;
`XOR`	efectuează operaţia logică `SAU EXCLUSIV` între elementul de imagine existent şi cel salvat.

4. SPRE O PROGRAMARE MODULARĂ ŞI STRUCTURATĂ

După cum am văzut în capitolul 2 orice algoritm logico-matematic poate fi astfel întocmit şi reprezentat încît să conţină numai trei tipuri de structuri de control fundamentale, bine definite: liniare, alternative şi repetitive, cu respectarea principalei caracteristici (reguli), de a avea o singură intrare şi o singură ieşire.

Din multiplele **facilităţi** oferite de utilizarea acestor structuri în descrierea algoritmilor, o **bună lizibilitate, reducerea considerabilă a posibilităţii de a comite erori, uşurinţa în urmărire, verificare** şi, eventual, **corectare, o eficienţă sporită** etc., subliniem, în special, **simplitatea cuplării şi imbricării** acestora în descriera algoritmilor complecşi. Datorită proprietăţii fundamentale (o singură intrare, o singură ieşire) se creează posibilitatea ca, indiferent de complexitatea unui algoritm, în ultimă instanţă, acesta să poată fi reprezentat prin cîteva structuri simple, conţinînd unul sau mai multe module.

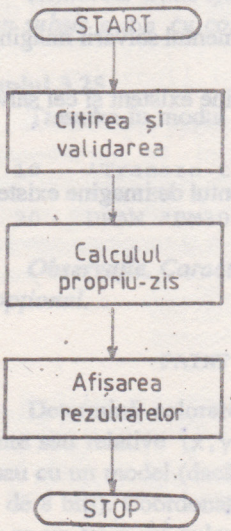


Fig. 4.1

În exemplele prezentate în capitolul 2 gruparea acţiunilor în module, deşi nu s-a subliniat în mod special, nu s-a efectuat la întîmplare ci pornind de la funcţiile pe care le îndeplinea grupul respectiv de acţiuni (operaţii) în cadrul algoritmului. Astfel în capitolul 2.4 algoritmul pentru calculul valorii funcţiei `f(x)`, (figura 2.15) cuprinde trei module: un modul de intrare (citirea valorii variabilei `x`), un modul de ieşire (afişarea soluţiei) şi modulul de calcul propriu-zis, module care, prin funcţiile lor, descriu în mod sintetic însăşi etapele şi modul de rezolvare al problemei analizate. În mod similar stau lucrurile şi în cazul exemplului 2.5 (figura 2.19).

Sigur, acestea sînt probleme foarte simple care pot fi rezolvate şi direct. Ne-am oprit însă asupra lor tocmai pentru a sublinia cea mai sintetică modalitate de desriere a unei probleme, care ar trebui să fie şi prima etapa în soluţionarea ei, şi anume **stabilirea funcţiilor generale şi a modulelor care să le îndeplinească**. Acestea ar fi (figura 4.1):

- **funcţia de citire** a valorilor variabilelor de intrare, efectuată de modulul conţinînd blocul (blocurile) de intrare şi eventualele validări;
- **funcţia de calcul propriu-zis**, efectuată de modulul (modulele) de calcul;
- **funcţia de afişare a soluţiilor**, efectuată de modulul (modulele) conţinînd blocurile de ieşire.

După cum vom vedea şi în continuare acestea sînt module universal valabile în descrierea oricărui algoritm. Diferenţieri apar abia în următoarele etape ale rezolvării problemei, etape în care,

pornind de la funcţia generală a fiecărui modul se trece la explicitarea într-un număr mai mare sau mai mic de 'submodule' interconectate conform specificului problemei de soluţionat.

În final, prin explicitarea şi interconectarea modulelor se obţine algoritmul (descris utilizînd schemele logice, în pseudocod sau direct într-un limbaj de programare) soluţionării problemei.

Cum în explicitarea modulelor s-a pornit de la general la particular, de la funcţia (funcţiile) ce urmează a fi îndeplinite şi prin faptul că modulele păstrează importanţa proprietate de a avea o singură intrare şi o singură ieşire, funcţie de tehnica şi experienţa programatorului, ele capătă un grad mai mic sau mai mare de generalitate, astfel încît pot fi utilizate ca atare şi în alte puncte ale algoritmului (programului), precum şi în soluţionarea altor probleme, ceea ce constituie un avantaj deosebit. O astfel de situaţie a fost întîlnită în exemplul 2.7 (capitolul 2), cînd, în descrierea algoritmului lui Euclid s-a utilizat procedura `SREST` realizată în exemplul 2.6, fapt evidenţiat în figurile 2.20 şi 2.21b.

Tehnica de realizare a *programelor modulare* poartă numele de **programare modulară** şi *urmăreşte reducerea complexităţii realizării şi testării programelor, prin descompunerea acestora în module interconectate în conformitate cu o ierarhie bine precizată, determinată de problema soluţionată.*

Modularizarea (realizarea de module program) se poate obţine şi fără utilizarea structurilor fundamentale prezentate în capitolul 2 dar, în acest caz, performanţele sînt mult mai modeste. Din această cauză optăm ca realizarea modulelor să se efectueze numai prin utilizarea structurilor de control fundamentale prezentate, conform principiilor programării structurate.

Programarea structurată este definită [13] *ca activitate ce urmăreşte elaborarea disciplinată a unui program (modul de program) pe baza unor structuri de control fundamentale şi date reprezentative problemei rezolvate, în scopul obţinerii unui produs program care reflectă clar algoritmul şi datele corespunzătoare problemei, este inteligibil, adaptabil şi testabil.*

Un prim pas spre realizarea de programe structurate şi modularizate a fost făcut pe parcursul capitolului 2, prin însăşi modul de abordare şi prezentare al modalităţilor de descriere ale algoritmilor prin utilizarea schemelor logice sau în pseudocod.

Din cele prezentate în capitolul 3 observăm însă că limbajul BASIC nu oferă facilităţi deosebite pentru realizarea unor astfel de programe. Totuşi, cu toate restricţiile impuse, realizarea unor programe structurate şi modularizate este posibilă.

Să examinăm cîteva exemple.

Exemplul 4.1

În capitolul 2, exemplul 2.7, a fost prezentat algoritmul lui Euclid pentru determinarea c.m.m.d.c. al două numere naturale diferite de zero. De data aceasta să determinăm c.m.m.d.c. fără a mai efectua împărţirea cu rest, ci prin scăderi succesive. Astfel, pentru a determina c.m.m.d.c. al numerelor N1 şi N2, acestea se compară între ele şi, dacă sînt egale, atunci c.m.m.d.c. este unul dintre ele, dacă nu, se scade cel mai mic din cel mai mare şi se reefectuează comparaţia; dacă au devenit egale, atunci acesta este c.m.m.d.c. al lor; dacă nu, se efectează o nouă scădere, şi aşa mai departe. Dacă, în final, se obţine identitatea 1=1 atunci numerele sînt prime între ele.

O variantă de program pentru determinarea în acest mod a c.m.m.d.c. al numerelor `N1` şi `N2` este prezentată în listingul 4.1 avînd schema logică în figura 4.2.

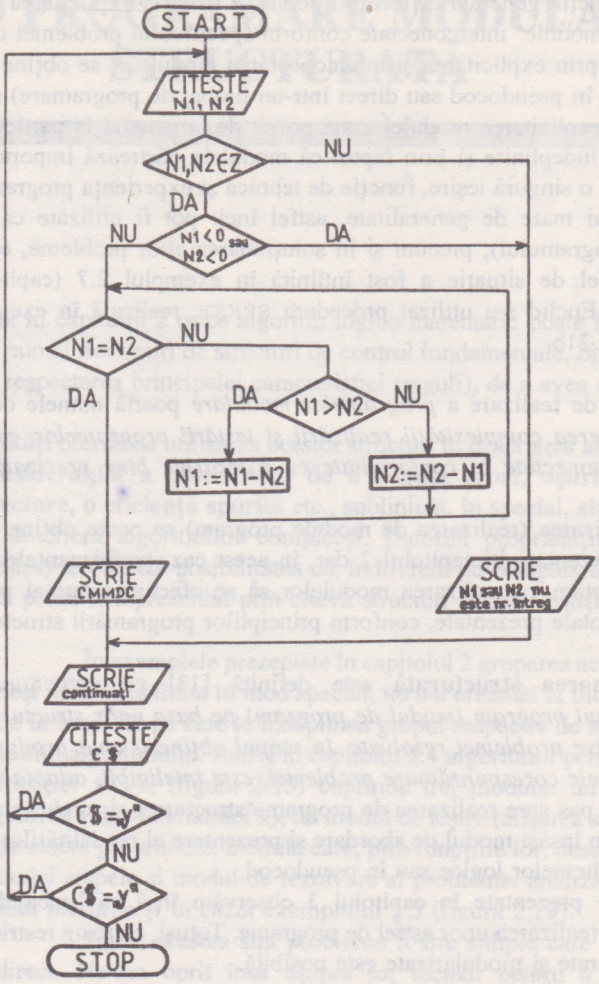


Fig. 4.2

```
10    PRINT "Programul determina C.M.M.D.C al N1 si N2"
20    INPUT "N1 = ",N1
30    INPUT "N2 = ",N2
40    IF INT(N1) <> N1 OR INT(N2) <> N2 THEN GOTO 110
50    IF N1 <= 0 OR N2 <= 0 THEN GOTO 110
60    IF N1 = N2 THEN GOTO 100
70    IF N1 > N2 THEN GOTO 90
80    N2=N2-N1 : GOTO 60
90    N1=N1-N2 : GOTO 60
100   PRINT "C.M.M.D.C. =";N1 : GOTO 120
110   PRINT "N1 sau N2 nu este numar natural pozitiv"
120   INPUT "Continuati ? (Y/N) :",C$
130   IF C$ = "Y" THEN GOTO 20
```

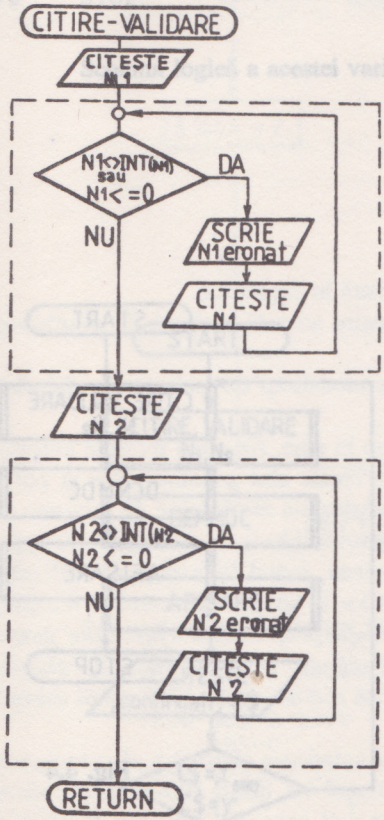
```
140   IF C$ = "y" THEN GOTO 20
150   STOP
```

Listing 4.2

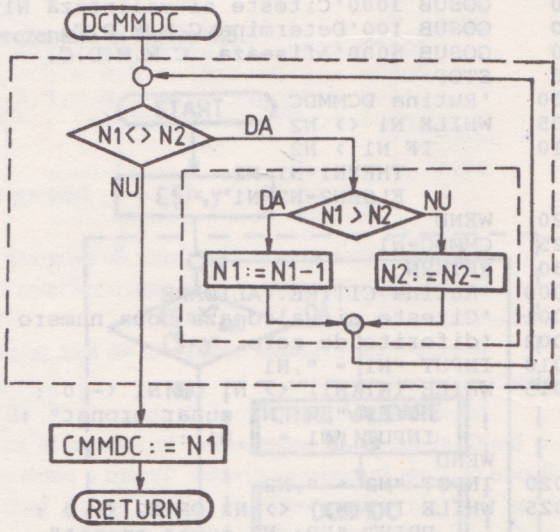
Deşi simplu, în această variantă de redactare, programul este oarecum dificil de urmărit.

Să rezolvăm acum aceeaşi problemă utilizind tehnicile programării structurate şi modulare. Prin program ne propunem să rezolvăm următoarele **funcţii:**

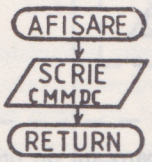
- **citirea şi validarea valorilor variabilelor de intrare;**
- **determinarea c.m.m.d.c.;**
- **afişarea c.m.m.d.c..**



a)



b)



c)

Fig. 4.3

Pentru realizarea acestor funcţii vom utiliza trei module de program:

- modulul CITIRE.VALIDARE (figura 4.3a) pentru citirea şi validarea valorilor variabilelor N1 şi N2; modulul conţine două structuri repetitive WHILE-DO, care determină repetarea introducerii valorilor fiecărei variabile atît timp cît acestea nu aparţin mulţimii valorilor admise (numerele naturale diferite de zero);

- modulul DCMMDC (figura 4.3b) pentru determinarea c.m.m.d.c. al numerelor N1 şi N2; modulul este alcătuit dintr-o structura WHILE-DO care conţine la rîndul ei o structură IF-THEN-ELSE;

- modulul AFISARE (figura 4.3c) pentru afişarea soluţiei; conţine instrucţiunea PRINT.

Cu aceste elemente schema logică devine cea din figura 4.4. Programul BASIC corespunzător (incluzînd şi modulele aferente) este prezentat în listingul 4.2.

```
10    PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11    PRINT "diferite de zero"
20    GOSUB 3000'Citeste si valineaza N1 si N2
30    GOSUB 100'Determina C.M.M.D.C.
40    GOSUB 5000'Afiseaza  C.M.M.D.C.
50    STOP
100   'Rutina DCMMDC
105   WHILE N1 <> N2
110       IF N1 > N2
            THENN1=N1-N2
            ELSEN2=N2-N1
120   WEND
125   CMMDC=N1
130   RETURN
3000  'Rutina CITIRE.VALIDARE
3001  'Citeste si valideaza doua numere intregi
3002  'diferite de zero
3010  INPUT "N1 = ",N1
3015  WHILE INT(N1) <> N1 OR N1 <= 0 :
          PRINT "ER: N1 nunar eronat" :
          INPUT "N1 = ",N1 :
      WEND
3020  INPUT "N2 = ",N2
3025  WHILE INT(N2) <> N2 OR N2 <= 0 :
          PRINT "ER: N2 numar eronat" :
          INPUT "N2 = ",N2 :
      WEND
3030  RETURN
5000  'Rutina AFISARE
5005  PRINT "C.M.M.D.C.=";CMMDC
5010  RETURN
```

Listing 4.2

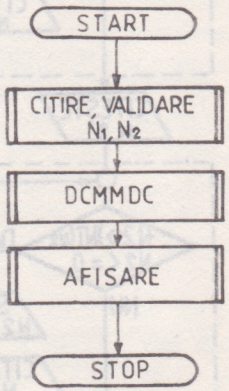


Fig. 4.4

Exemplul 4.2

Avînd o secvenţă oarecare de program, de exemplu determinarea c.m.m.d.c. prezentată în exemplul anterior, să întocmim un program care să permită reluarea opţională a execuţiei acesteia.

Prezentăm două variante de soluţionare a acestei probleme.

O primă variantă este cea ilustrată prin programul din listingul 4.3 şi constă în inserarea după secvenţa de program propriu-zisă (liniile 20-40) a două instrucţiuni: o instrucţiune INPUT (linia 50)

prin care se solicită şi se primeşte un răspuns privind opţiunea de continuare, şi o instrucţiune IF-GOTO (linia 60) prin care, în urma evaluării expresiei logice, determină, fie salt la linia 20 şi reluarea secvenţei pentru noi valori ale lui N1 şi N2, fie trecerea la linia 70 (STOP).

```
 1    'Varianta de executie repetata
 2    'a unei secvente de program
10    PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11    PRINT "diferite de zero"
20    GOSUB 3000'Citeste si valideaza N1 si N2
30    GOSUB 100'Determina C.M.M.D.C.
40    GOSUB 5000'Afiseaza C.M.M.D.C.
50    INPUT "Continuati ? (Y/N) : ",C$
60    IF C$ = "Y" OR C$ = "y" GOTO 20
70    STOP
```

Listing 4.3

Schema logică a acestei variante este prezentată în figura 4.5.

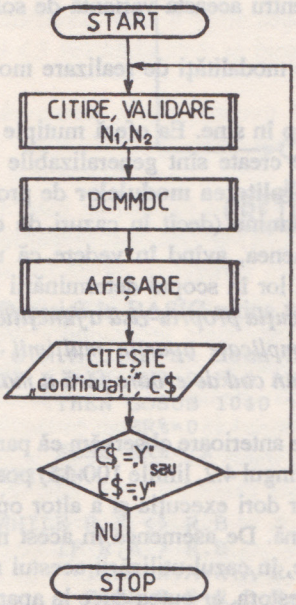


Fig. 4.5

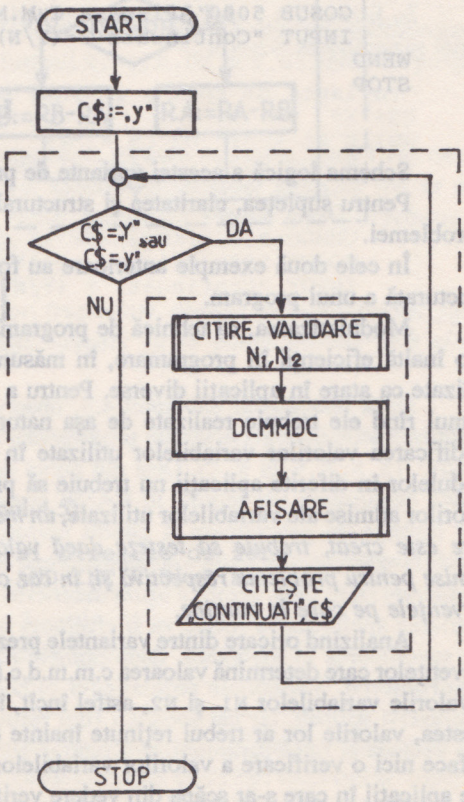


Fig. 4.6

O altă variantă de soluţionare a problemei este includerea secvenţei de executat într-o structură de tip WHILE-DO (listingul 4.4). În acest caz, deoarece structura WHILE-DO presupune întîi evaluarea expresiei logice (condiţiei) şi apoi, dacă aceasta este adevărată, parcurgerea secvenţei, instrucţiunea WHILE (linia 30) este necesar a fi precedată de o instrucţiune prin care să se pregătească evaluarea în sensul dorit a expresiei logice respective (în exemplul prezentat în listingul 4.4 acesta se realizează prin instrucţiunea de atribuire din linia 20). După intrarea în ciclu pregătirea evaluării expresiei logice a instrucţiunii WHILE se efectuează în linia 70 prin instrucţiunea INPUT. Funcţie de valoarea primită de variabila C$, secvenţa se poate relua sau nu.

```
 1    'Varianta structurata de executie repetata
 2    'a unei secvente de program
10    PRINT "Determinarea C.M.M.D.C. al doua numere naturale"
11    PRINT "diferite de zero"
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40        GOSUB 3000'Citire si validare N1 si N2
50        GOSUB 100'Determina C.M.M.D.C.
60        GOSUB 5000'Afiseaza C.M.M.D.C.
70        INPUT "Continuati ? (Y/N) : ",C$
80    WEND
90    STOP
```

Listing 4.4

Schema logică a acestei variante de program este prezentată în figura 4.6.

Pentru supleţea, claritatea şi structuralitatea ei, optăm pentru această variantă de soluţionare a problemei.

În cele două exemple anterioare au fost prezentate unele modalităţi de realizare modulară şi structurată a unui program.

Modularizarea, ca tehnică de programare, nu este un scop în sine. Ea oferă mutiple avantaje şi o înaltă eficienţă în programare, în măsura în care modulele create sînt generalizabile şi pot fi utilizate ca atare în aplicaţii diverse. Pentru a putea asigura **generalitatea modulelor** de program, în primul rînd ele trebuie realizate de aşa natură, încît să nu determine (decît în cazuri de excepţie) modificarea valorilor variabilelor utilizate în program; de asemenea, avînd în vedere că utilizarea modulelor în diferite aplicaţii nu trebuie să presupună şi analiza lor în scopul determinării mulţimii valorilor admise ale variabilelor utilizate, *un modul, înainte de execuţia propriu-zisă a funcţiilor pentru care este creat, trebuie să testeze dacă valorile variabilelor implicate aparţin mulţimii valorilor admise pentru problema respectivă şi, în caz că nu, să returneze un cod de eroare fără a mai executa secvenţele pe care le conţine.*

Analizînd oricare dintre variantele prezentate în exemplele anterioare observăm că parcurgerea secvenţelor care determină valoarea c.m.m.d.c.(rutina DCMMDC, listingul 4.2, liniile 100-15) poate afecta şi valorile variabilelor N1 şi N2, astfel încît, în cazul în care s-ar dori execuţia şi a altor operaţii cu acestea, valorile lor ar trebui reţinute înainte de saltul la subrutină. De asemenea în acest modul nu se face nici o verificare a valorilor variabilelor N1 şi N2, ceea ce, în cazul utilizării acestui modul în alte aplicaţii în care s-ar scăpa din vedere verificarea valorilor acestora, ar putea duce la apariţia unor erori.

Cum am putea realiza un modul (rutină) cu un grad cît mai mare de generalitate ?

Exemplul 4.3

Să se generalizeze rutina de determinare a c.m.m.d.c. utilizată în exemplele anterioare.

Pentru aceasta vom utiliza variabilele R.A, R.B şi R.C ca variabile interne pentru determinarea c.m.m.d.c., precum şi variabila ER% pentru transmiterea codului de eroare la întoarcerea din rutină. Dacă execuţia rutinei a decurs normal ER% va conţine valorea 0, în caz contrar conţinînd valoarea -2.

Schema logică a rutinei DCMMDC devine (figura 4.7):

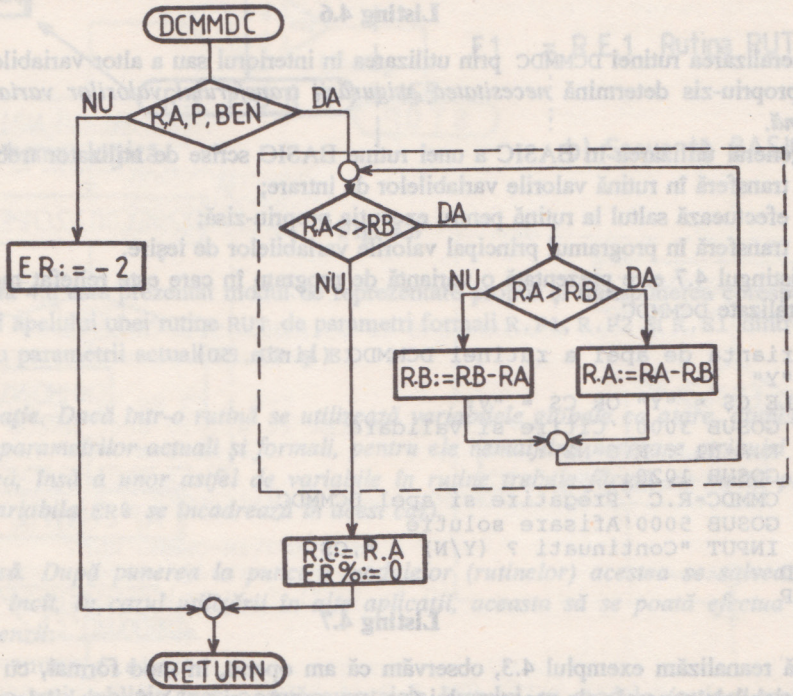


Fig. 4.7

Descrisă în BASIC rutina DCMMDC este (listingul 4.5):

```
1020  'C.M.M.D.C. al doua numere naturale si diferite de zero
1025  IF R.A>0 AND R.B>0 AND INT(R.A)=R.A AND INT(R.B)=R.B
          THEN GOSUB 1040 :
               ER%=0
        ELSE ER%=-2
1030  RETURN
1035  '
1040  WHILE R.A <> R.B
1045      IF R.A > R.B
            THEN R.A=R.A-R.B
            ELSE R.B=R.B-R.A
1050  WEND
1055  R.C=R.A
1060  RETURN
```

Listing 4.5

Utilizarea unei variabile de eroare impune refacerea şi a rutinei de afişare, care acum devine (listingul 4.6):

```
5000  'Rutina AFISARE
5005  IF ER% = 0
          THEN PRINT "C.M.M.D.C.=";CMMDC
          ELSE PRINT "ER = ";ER%
5010  RETURN
```

Listing 4.6

Generalizarea rutinei DCMMDC prin utilizarea în interiorul sau a altor variabile decît cele din programul propriu-zis determină *necesitatea asigurării transferului valorilor variabilelor spre şi dinspre rutină.*

În general utilizarea în BASIC a unei rutine BASIC scrise de utilizator trebuie să decurgă astfel: - se transferă în rutină valorile variabilelor de intrare;
- se efectuează saltul la rutină pentru execuţia propriu-zisă;
- se transferă în programul principal valorile variabilelor de ieşire.

În listingul 4.7 este prezentată o variantă de program în care este reliefat modul de apel al rutinei generalizate DCMMDC.

```
10    'Varianta de apel a rutinei DCMMDC (linia 50)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40        GOSUB 3000 'Citire si validare
50        R.A=N1 : R.B=N2 :
          GOSUB 1020:
          CMMDC=R.C 'Pregatire si apel DCMMDC
60        GOSUB 5000'Afisare solutie
70        INPUT "Continuati ? (Y/N) : ",C$
80    WEND
90    STOP
```

Listing 4.7

Dacă reanalizăm exemplul 4.3, observăm că am operat, în mod formal, cu două tipuri de variabile: variabilele generale ale programului (N1, N2, CMMDC şi ER%) şi variabilele specifice rutinei DCMMDC (R.A, R.B şi R.C). Deşi din punct de vedere al majorităţii variantelor de BASIC nu există această departajare (de regulă, în BASIC toate variabilele sînt globale), spre a înlesni însuşirea şi a altor limbaje de programare şi pentru uniformitate, vom numi variabilele programului propriu-zis **variabile globale**, iar cele utilizate numai în rutine **variabile locale**. De asemenea, din acelaşi considerent şi, în plus, pentru a uşura descrierea algoritmilor utilizînd schemele logice sau pseudocodul, descrierea apelurilor la rutine va fi efectuată utilizînd blocul de procedură prezentat în capitolul 2.2, prin specificarea în paranteză, după numele rutinei, a variabilelor implicate, într-o listă numită **lista de parametrii**. Variabilele programului principal care apar în lista parametrilor de apel se numesc **parametrii actuali**, iar cei specifici (din rutină) corespunzători lor, se numesc **parametrii formali**. Între ordinea şi tipul parametrilor actuali şi formali trebuie să fie o corespondenţă biunivocă.

Observaţie importantă. *Cum în majoritatea variantelor de BASIC nu există nici o distincţie între variabilele globale şi locale, transferul de valori între parametrii actuali şi formali utilizaţi nu se efectuează automat şi, ca atare, în program trebuie să apară efectiv operaţiile de atribuire respective.*

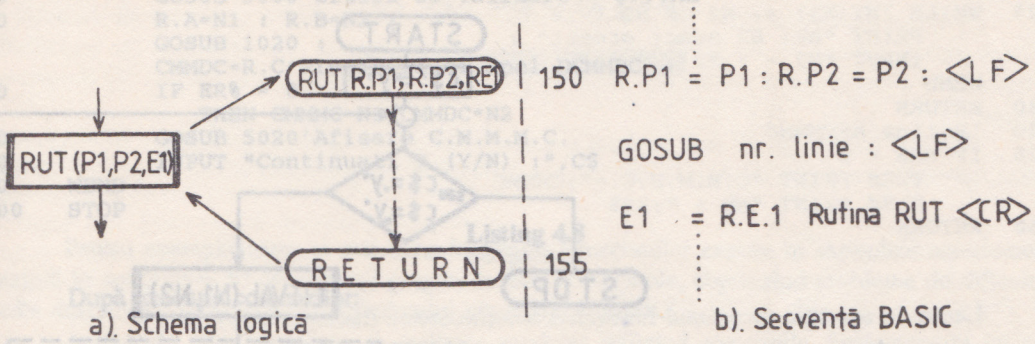


Fig. 4.8

În figura 4.8 este prezentat modul de reprezentare propus şi transpunerea corespunzătoare în BASIC în cazul apelului unei rutine `RUT` de parametri formali `R.P1`, `R.P2` şi `R.E1` dintr-un program oarecare, pentru parametrii actuali `P1`, `P2` şi `E1`.

Observaţie. Dacă într-o rutină se utilizează variabilele globale ca atare, atunci ele nu mai apar în listele parametrilor actuali şi formali, pentru ele nemaifiind necesare atribuiri (nu ar avea sens). Utilizarea, însă a unor astfel de variabile în rutine trebuie făcută cu multă precauţie (În exemplul 4.3 variabila `ER%` se încadrează în acest caz).

***Remarcă.** După punerea la punct a modulelor (rutinelor) acestea se salvează în fişiere separate astfel încît, în cazul utilizării în alte aplicaţii, aceasta să se poată efectua prin simpla execuţie a comenzii:*

```
MERGE "nume_fisier".
```

Pentru a mai sublinia încă o dată avantajele lucrului cu module (rutine) mai prezentăm următorul exemplu.

Exemplul 4.4

Să se întocmească un program pentru determinarea c.m.m.m.c al două numere naturale diferite de zero.

Programul trebuie să îndeplinească următoarele funcţii:

- citirea şi validarea a două numere naturale diferite de zero;
- determinarea c.m.m.m.c.;
- afişarea rezultatului;
- reluarea opţională a secvenţei de program pentru alte valori ale variabilelor de intrare.

Observăm că prima funcţie o avem deja programată în rutina `CITIRE.VALIDARE` (listingul 4.2, liniile 3000-3030), rutina pe care o presupunem salvată în fişierul `CITVAL`.

Schema logică de determinare a c.m.m.m.c al două numere naturale diferite de zero este cea din figura 4.9. Avînd deja rutina `DCMMDC` pentru determinarea c.m.m.d.c. realizarea acestei funcţii este extrem de simplă.

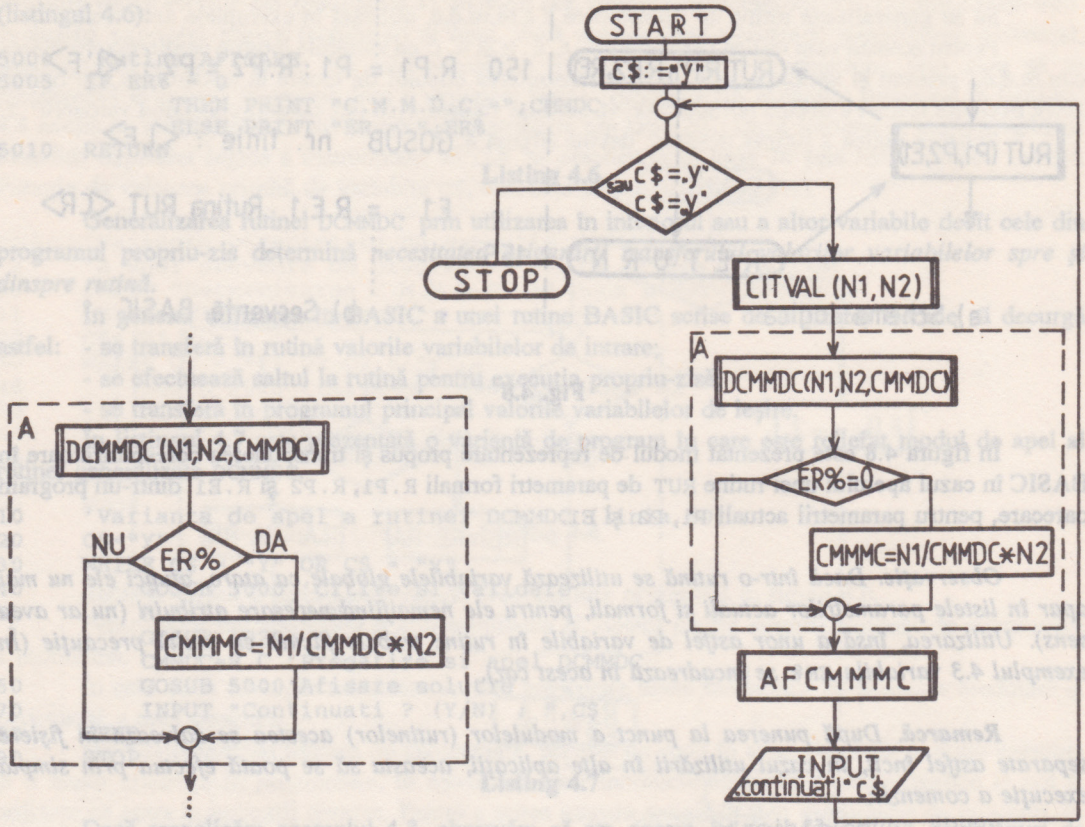


Fig. 4.9 **Fig. 4.10**

Pentru afişarea soluţiei folosim o rutină asemănătoare celei prezentate în exemplul 4.3.

În vederea asigurării posibilităţii de reluare opţională a execuţiei secvenţei dorite vom utiliza structura `WHILE-DO` descrisă în exemplul 4.2.

Schema logică a programului este cea din figura 4.10. În blocul `A` (încadrat în figura 4.10 cu linie întreruptă) regăsim algoritmul de determinare a c.m.m.m.c. din figura 4.9. Practic, cu o foarte mică excepţie (o modificare nesemnificativă în rutina de afişare), aceasta este singura parte nouă faţă de cele prezentate în exemplul 4.3, necesară pentru soluţionarea problemei propuse în acest exemplu.

Programul propriu-zis este cel prezentat în listingul 4.8.

```
10    PRINT "Determinarea C.M.M.M.C. al doua numere naturale"
11    PRINT "diferite de zero "
20    C$="Y"
```

```
30    WHILE C$ = "Y" OR C$ = "y"
40          GOSUB 3000'Citire si validare - CITVAL -
50          R.A=N1 : R.B=N2 :
            GOSUB 1020 :
            CMMDC=R.C'Pregatire si apel DCMMDC
60          IF ER% = 0
               THEN CMMMC=N1/CMMDC*N2
70          GOSUB 5020'Afisare C.M.M.M.C.
80          INPUT "Continuati ? (Y/N) :",C$
90    WEND
100   STOP
```

Listing 4.8

După execuţia comenzilor:

```
MERGE "CITVAL"
MERGE "DCMMDC"
MERGE "AFCMMMC"
```

acesta devine (listigul 4.9):

```
10    PRINT "Determinarea C.M.M.M.C. al doua numere naturale"
11    PRINT "diferite de zero "
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40          GOSUB 3000'Citire si validare - CITVAL -
50          R.A=N1 : R.B=N2 :
            GOSUB 1020 :
            CMMDC=R.C'Pregatire si apel DCMMDC
60          IF ER% = 0
               THEN CMMMC=N1/CMMDC*N2
70          GOSUB 5020'Afisare C.M.M.M.C.
80          INPUT "Continuati ? (Y/N) :",C$
90    WEND
100   STOP
1020  'C.M.M.D.C. al doua numere naturale si diferite de zero
1025  IF R.A>0 AND R.B>0 AND INT(R.A)=R.A AND INT(R.B)=R.B
         THEN GOSUB 1040 :
              ER%=0
         ELSE ER%=-2
1030  RETURN
1035  '
1040  WHILE R.A <> R.B
1045        IF R.A > R.B
              THEN R.A=R.A-R.B
              ELSE R.B=R.B-R.A
1050  WEND
1055  R.C=R.A
1060  RETURN
3000  'Rutina CITIRE.VALIDARE
3001  'Citeste si valideaza doua numere intregi
3002  'diferite de zero
3010  INPUT "N1 = ",N1
3015  WHILE INT(N1) <> N1 OR N1 <= 0 :
         PRINT "ER: N1 nunar eronat" :
         INPUT "N1 = ",N1 :
```

```
      WEND
3020  INPUT "N2 = ",N2
3025  WHILE INT(N2) <> N2 OR N2 <= 0 :
          PRINT "ER: N2 numar eronat" :
          INPUT "N2 = ",N2 :
      WEND
3030  RETURN
5020  'Rutina AFCMMMC
5025  IF ER% = 0
          THEN PRINT "C.M.M.M.C.=";CMMMC
          ELSE PRINT "ER = ";ER%
5030  RETURN
```

Listing 4.9

Lansat în execuţie programul furnizează soluţii corecte (listingul 4.10).

```
RUN
Determinarea C.M.M.M.C. al doua numere naturale
diferite de zero
N1 = 4
N2 = 6
C.M.M.M.C.= 12
Continuati ? (Y/N) :Y
N1 = -1
ER: N1 numar eronat
N1 = 1
N2 = 8
C.M.M.M.C.= 8
Continuati ? (Y/N) :Y
N1 = 24
N2 = 0
ER: N2 numar eronat
N2 = 63
C.M.M.M.C.= 504
Continuati ? (Y/N) :N
Break in 100
Ok
```

Listing 4.10

Analizînd exemplul 4.4 observăm că pentru soluţionarea problemei a fost necesară scrierea numai a cîtorva linii noi, în rest bazîndu-ne pe utilizarea unor module create anterior.

Formînd treptat o bibliotecă de astfel de module se poate ajunge la performanţa de a scrie rapid, corect şi cu minim de efort programe din ce în ce mai complexe.

5. APLICAŢII

Pentru exemplificarea şi mai buna înţelegere a noţiunilor expuse în capitolele anterioare se prezintă în continuare 40 de aplicaţii grupate în şase subcapitole, cuprinzînd probleme de dificultate medie desprinse din tematica învăţămîntului gimnazial şi liceal.

Aplicaţiile sînt concepute şi realizate modular utilizînd structurile fundamentale cu care operează programarea structurată, astfel încît, pe lîngă liniile program specifice fiecăreia (cu etichete de linii între 5 şi 999), au fost definite şi utilizate module generale de calcul (cu etichete de linii între 1000 şi 2999), de citire şi validare a unor date (cu etichete de linii între 3000 şi 4999) şi de afişare a rezultatelor (avînd etichete de linii în intervalul 5000 la 5999).

Necesitatea unui cît mai eficient control al modului de desfăşurare al diferitelor rutine utilizate în construcţiile aplicaţiilor prezentate în acest capitol a impus definirea variabilei `ER%` care conţine în orice moment codul ultimei erori depistate. Definirea modulelor s-a efectuat o singură dată, în aplicaţiile care le utilizează prima dată. Funcţiile şi principalele caracteristici ale modulelor utilizate (denumire simbolică, parametri de intrare şi / sau ieşire, variabilele "globale" utilizate, codurile de eroare returnate etc.) sînt prezentate în anexa A.

Mulţimile valorilor etichetelor liniilor modulelor fiind disjuncte, ele se pot utiliza în programe şi în alte combinaţii decît cele prezentate (cu respectarea însă a condiţiilor precizate în anexa 1).

Structurile generale (programele principale) ale aplicaţiilor 1 - 38 respectă, cu unele particularităţi, structura de program prezentată în figura 5.1.

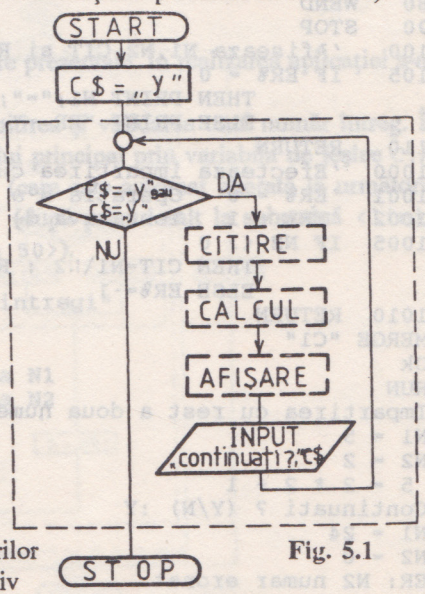


Fig. 5.1

După cum se poate observa cu uşurinţă este vorba de o structură `WHILE-DO`, structură prezentată pe larg în 3.8.1. S-a ales acest mod de abordare deoarece oferă utilizatorului posibilitatea de a rămîne în conversaţie cu calculatorul atît timp cît doreşte; pot fi studiate astfel diferite aspecte generale sau particulare ale problemelor modelate fără a mai fi necesară relansarea programelor pentru fiecare set de date particular.

Pentru separarea, pe cît posibil, a variabilelor specifice modulelor de cele cu care operează programele propriu-zise, identificatorii acestora (consideraţi din punct de vedere logic 'locali') sînt precedaţi de `R.`, `C.` şi respectiv `S.`. Aceasta impune, de regulă, ca la utilizare să se efectueze explicit transferul valorilor parametrilor implicaţi, înainte şi / sau după apelarea modulelor. Deşi relativ incomodă, această modalitate de realizare a modulelor le conferă acestora un grad relativ mare de

generalitate şi o oarecare independenţă faţă de programele apelante.

Reunirea modulelor într-o bibliotecă şi dezvoltarea treptată a sa, se poate dovedi deosebit de eficientă pentru utilizatorul BASIC, înlăturînd parţial inconvenientele datorate lipsei (în majoritatea covîrşitoare a generaţiilor şi versiunilor BASIC) procedurilor întîlnite în alte limbaje de progamare evoluate.

Pentru a conferi un cît mai profund caracter practic şi a înlesni parcurgerea, înţelegerea şi analiza aplicaţiilor prezentate, acestea sînt însoţite de rezultatele execuţiilor efectuate cu date de test.

5.1 Numere întregi, baze de numeraţie

● Împărţire cu rest a două numere naturale

Aplicaţia, prezentată în listingul 5.1, determină cîtul `CIT` şi restul `REST` ale împărţirii întregi a numerelor `N1` şi `N2`.

```
10    PRINT "Impartirea cu rest a doua numere naturale nenule"
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3000 'Citeste si valideaza N1 si N2
50     GOSUB 1000 'Efectueaza impartirea cu rest
60     GOSUB 100 'Afiseaza N1,N2,CIT si REST
70    INPUT "Continuati ? (Y/N) :"
      C$
80    WEND
90    STOP
100    'Afiseaza N1,N2,CIT si REST
105    IF ER% = 0
            THEN PRINT N1;"=";N2"*";CIT;"+";REST
            ELSE PRINT "ER =";ER%;       " Tentativa de impartire la zero"
110    RETURN
1000   'Efecteaza impartirea cu rest N1=N2*CIT+REST
1001   'ER% = 0   Operatia s-a efectuat corect
1002   'ER% =-1   Tentativa de impartire la zero
1005   IF N2 <> 0
            THEN CIT=N1\N2 : REST N1 MOD N2 : ER%=0
            ELSE ER%=-1
1010   RETURN
MERGE "C1"
Ok
RUN
Impartirea cu rest a doua numere naturale nenule
N1 = 5
N2 = 2
 5 = 2 * 2 + 1
Continuati ? (Y/N) :Y
N1 = 24
N2 = 0
ER: N2 numar eronat
N2 = 6
```

```
 24 = 6 * 4 + 0
Continuati ? (Y/N):N
Break in 90
Ok
```

Listing 5.1

Citirea şi validarea deîmpărţitului N1 şi împărţitorului N2 se realizează în rutina C1 (anexa A), prezentată în capitolul 4, figura 4.3.a, listingul 4.9. Prin modul de realizare, aceasta obligă utilizatorul să tasteze numai numere corecte din punctul de vedere al problemei date.

Determinarea valorilor variabilelor CIT şi REST este asigurată prin rutina R1, listingul 5.1, liniile 1000-1010. Rutina verifică valoarea împărţitorului şi, în cazul în care acesta este 0, returnează valoarea ER%=-1 (tentativa de împărţire la 0), fără a mai efectua operaţia. Schema logică a rutinei R1 este prezentată în figura 5.2. Rutina nu afectează valorile variabilelor N1 şi N2.

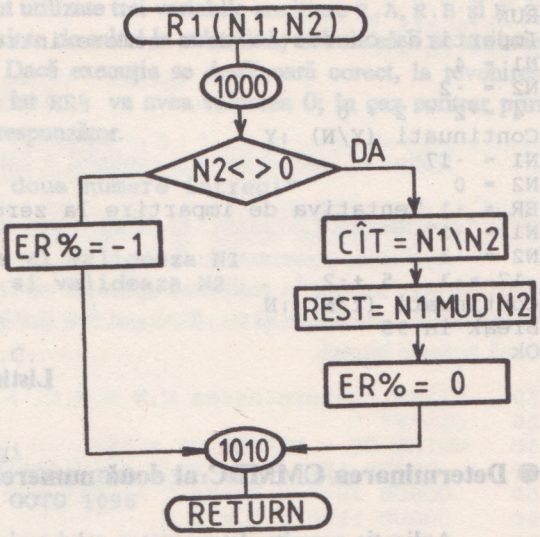


Fig. 5.2

● Împărţirea cu rest a două numere întregi

Această aplicaţie este o generalizare a celei anterioare prezentate. În realizarea aplicaţiei s-au utilizat rutinele R1 (anexa A) şi C2.

Rutina C2, listingul 5.2, liniile 3040-3055, asigură citirea şi validarea unui număr întreg. În cazul tastării unui număr corect, acesta este oferit programului principal prin variabila de ieşire C.A. Deoarece aceasta este o variabilă de manevră a rutinei C2 (care este automat alterată la următorul salt la subrutină), valoarea ei este atribuită variabilelor N1 (după primul salt la subrutină <linia 40>), respectiv N2 (după al doilea salt la subrutină <linia 50>).

```
10   PRINT "Impartirea cu rest a doua numere intregi"
20   C$="Y"
30   WHILE C$ = "Y" OR C$ = "y"
40    GOSUB 3040 : N1=C.A'Citeste si valideaza N1
50    GOSUB 3040 : N2=C.A'Citeste si valideaza N2
60    GOSUB 1000'Efectueaza impartirea
70    GOSUB 100'Afisare
80    INPUT "Continuati ? (Y/N) :",C$
90   WEND
95   STOP
3040  'Citeste si valideaza un numar intreg
3045  INPUT "N = ",C.A
3050    WHILE C.A <> INT(C.A) :
```

```
     PRINT "ER: numar eronat" :
     INPUT "N = ",C.A :
     WEND
3055  RETURN
MERGE "R1"
Ok
RUN
Impartirea cu rest a doua numere intregi
N1 = 4
N2 = -2
 4 =-2 *-2 + 0
Continuati (Y/N) :Y
N1 = -17
N2 = 0
ER = -1 Tentativa de impartire la zero
N1 = -17
N2 = -3
-17 =-3 * 5 +-2
Continuati (Y/N) :N
Break in 95
Ok
```

Listing 5.2

● Determinarea CMMDC al două numere întregi

Aplicaţia permite determinarea celui mai mare divizor comun al două numere întregi şi este utilă, alături de aplicaţiile următoare, mai ales elevilor din clasele 5 - 7 în lucrul cu fracţii.

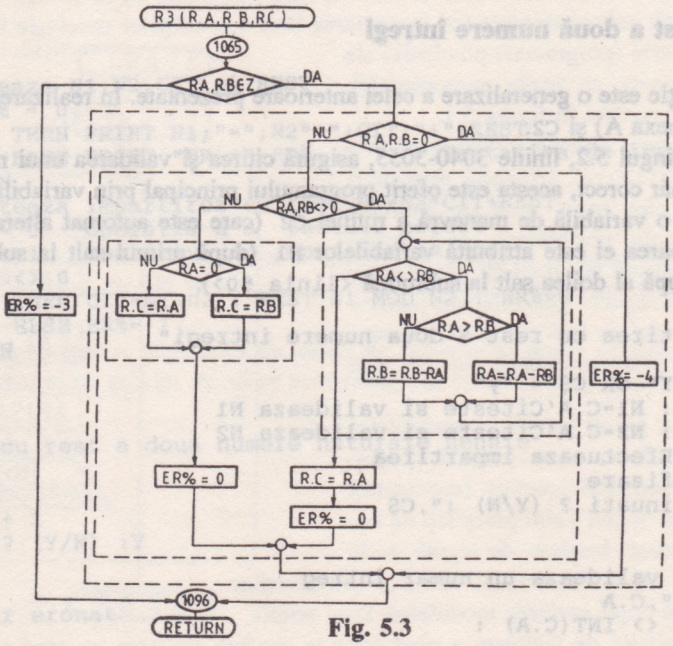


Fig. 5.3

În realizarea aplicaţiei au fost utilizate rutinele C2 (anexa A), pentru introducerea şi validarea variabilelor de intrare, R3 pentru determinarea CMMDC al două numere întregi şi S1 pentru afişarea soluţiei.

Rutina R3 este prezentată în listingul 5.3, liniile 1065 - 1096. Schema logică a rutinei este ilustrată în figura 5.3. La determinarea CMMDC sînt utilizate trei variabile auxiliare R.A, R.B şi R.C. Variabilelor R.A şi R.B, considerate de intrare înainte de saltul la subrutină, trebuie să li se atribuie valorile variabilelor al căror CMMDC îl căutăm. Dacă execuţia se desfăşoară corect, la revenirea din subrutină variabila R.C va conţine CMMDC, iar ER% va avea valoarea 0; în caz contrar prin variabila ER% se va transmite un cod de eroare corespunzător.

```
10    PRINT "Determinarea C.M.M.D.C. al doua numere intregi"
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3040:R.A=ABS(C.A)  'Citeste si valideaza N1
50     GOSUB 3040:R.B=ABS(C.A) 'Citeste si valideaza N2
60     GOSUB 1065    'C.M.M.D.C.
70     IF ER% = 0 THEN CMMDC=R.C
80     GOSUB 5000       'Afisare C.M.M.D.C.
90     INPUT "Continuati ? (Y/N) :",C$
100   WEND
110   STOP
1065   'C.M.M.D.C. al doua numere intregi
1070   IF R.A<>INT(R.A) OR R.B<>INT(R.B) THEN ER%=-3 : GOTO 1096
1071   IF R.A=0 AND R.B=0 THEN ER%=-4 : GOTO 1096
1075   IF R.A<>0 AND R.B<>0
       THENGOSUB 1087
       ELSEGOSUB 1094
1080   RETURN
1086   '
1087   WHILE R.A<>R.B
1088     IF R.A > R.B
              THENR.A=R.A-R.B
              ELSER.B=R.B-R.A
1089   WEND
1090   R.C=R.A
1091   ER%=0
1092   RETURN
1093   '
1094   IF R.A = 0
              THENR.C=R.B
              ELSER.C=R.A
1095   ER%=0
1096   RETURN
MERGE "C2"
Ok
MERGE "S1"
Ok
RUN
Determinarea C.M.M.D.C. al doua numere intregi
N = 24
N = 18
C.M.M.D.C.= 6
Continuati ? (Y/N) :Y
N = 0
```

```
N = 0
ER = -4
Continuati ? (Y/N) :Y
N = -12
N = -15
C.M.M.D.C.= 3
Continuati ? (Y/N) :N
Break in 110
Ok
```

Listing 5.3

Rutina `S1` este definită în capitolul 4, listingul 4.6.

● Determinarea CMMMC al două numere întregi

Aplicația, prezentată în listingul 5.4, permite determinarea celui mai mic multiplu comun al două numere întregi. În realizarea aplicației au fost utilizate rutinele `C2`, `R3`, `S3` (anexa A), definite în aplicațiile anterioare, și `R4` (listingul 5.4, liniile 1100 - 1115) care determină efectiv CMMMC al două numere întregi.

```
10    PRINT "Determinarea C.M.M.M.C. al doua numere intregi"
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40      GOSUB 3040:R.A=ABS(C.A)'Citeste si valideaza N1
50      GOSUB 3040:R.B=ABS(C.A)'Citeste si valideaza N2
60      GOSUB 1100'C.M.M.M.C.
70      IF ER% = 0 THEN CMMMC=R.C
80      GOSUB 5020'Afisare C.M.M.M.C.
90      INPUT "Continuati ? (Y/N) :",C$
100   WEND
110   STOP
1100   'C.M.M.M.C. al doua numere intregi
1105   R.A1=R.A: R.B1=R.B : GOSUB 1065 'C.M.-
M.D.C.
1110   IF ER% = 0 THEN R.C=R.A1/R.C*R.B1
1115   RETURN
MERGE "C2"
Ok
MERGE "R3"
Ok
MERGE "S2"
Ok
RUN
Determinarea C.M.M.M.C. al doua numere intregi
N = 6
N = 4
C.M.M.M.C.= 12
Continuati ? (Y/N) :Y
N = 84
N = 20
C.M.M.M.C.= 420
Continuati ? (Y/N) :N
Break in 110
Ok
```

Listing 5.4

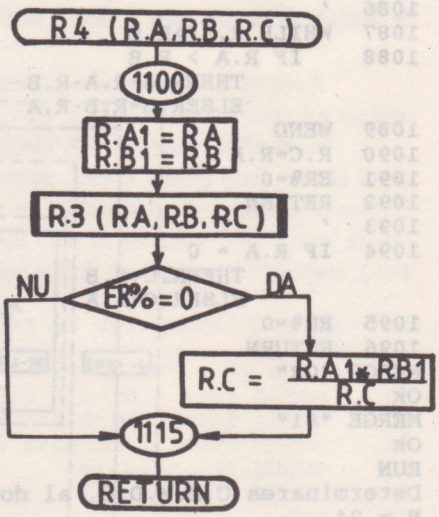


Fig. 5.4

Rutina R4, a cărei schemă logică este prezentată în figura 5.4, utilizează variabilele de intrare R.A şi R.B, de ieşire R.C (conţine valoarea CMMMC), precun şi variabilele auxiliare R.A1 şi R.A2 cu rol de a 'proteja' (de a păstra nemodificate) valorile variabilelor R.A şi R.B utilizate şi de rutina R3 apelată de aceasta.

● Determinarea CMMDC al maxim 20 de numere întregi

Aplicaţia constituie o generalizare a celor prezentate în 5.1.3 pentru un şir de maxim 20 de numere întregi. În cadrul acestei aplicaţii se definesc două noi rutine C3 şi R5.

Rutina C3, listingul 5.5, liniile 3060 - 3095, permite citirea şi validarea unui vector de maxim 20 de numere întregi. Valorile celor N numere corect tastate sînt depuse succesiv în vectorul A. Variabilele A şi N sînt considerate globale.

```
10    PRINT "Determinarea C.M.M.D.C. al 2<=N<=20 numere intregi"
15    DIM A(20)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3060'Citeste N si A(N)
50     R.N=N : GOSUB 1120  'C.M.M.D.C
60     IF ER% = 0 THEN CMMDC=R.C
70     GOSUB 5000'Afisare C.M.M.D.C
80     INPUT "Continuati ? (Y/N) :",C$
90    WEND
100   STOP
1120   'C.M.M.D.C AL 2<=N<=20 numere intregi
1125   IF R.N<2 OR R.N>20 THEN ER%=-5 : GOTO 1135
1126   R.E=0 : R.I=2
1127   WHILE R.E = 0 AND R.I <= R.N :
       R.E=A(R.I-1) :
       R.I=R.I+1
1128   WEND
1129   R.I=R.I-1
1130   IF R.E <> 0
            THEN R.A=ABS(A(R.I-1)) :
            R.B=ABS(A(0)) :
            GOSUB 1065 :
            R.I=2 :
            WHILE ER%=0 AND R.C<>1 AND R.I<=R.N :
       R.A=R.C :
       R.B=ABS(A(R.I-1)) :
       GOSUB 1065 :
       R.I=R.I+1 :
            WEND
       ELSE ER%=-4
1135   RETURN
3060   'Citeste 2<=N<=20 numere intregi in A(N)
3065   INPUT "Precizati N (2<=N<=20) : ",N
3066   WHILE INT(N) <> N OR N < 2 OR N > 20 :
          PRINT "ER : numar eronat" :
          INPUT "Precizati N (2<=N<=20) : ",N :
       WEND
3075   FOR C.I = 0 TO N-1
```

```
3080      PRINT "A(";C.I;") ";:
            INPUT "= ",A(C.I)
3085    WHILE INT(A(C.I))<> A(C.I):
          PRINT "ER : numar eronat":
          PRINT "A(";C.I;") " ;
            INPUT "= ",A(C.I) :
        WEND
3090  NEXT
3095  RETURN
MERGE "R3"
Ok
MERGE "S1"
Ok
RUN
Determinarea C.M.M.D.C. al 2<=N<=20
numere intregi
Precizati N (2<=N<=20) : 3
A( 0 ) = 0
A( 1 ) = 12
A( 2 ) = 6
C.M.M.D.C.= 6
Continuati ? (Y/N) :Y
Precizati N (2<=N<=20) : 6
A( 0 ) = 21
A( 1 ) = 63
A( 2 ) = 147
A( 3 ) = 231
A( 4 ) = 105
A( 5 ) = 273
C.M.M.D.C.= 21
Continuati ? (Y/N) :N
Break in 100
Ok
```

Listing 5.5

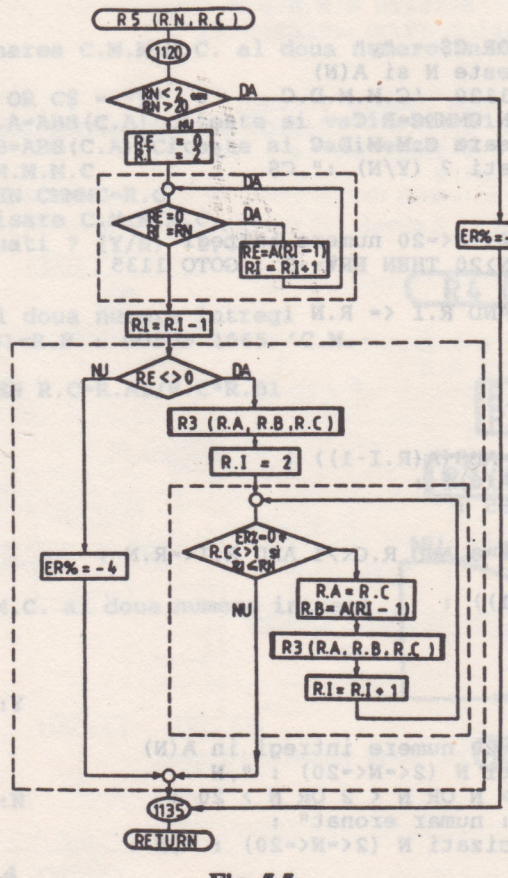


Fig. 5.5

Rutina R5, listingul 5.5, liniile 1120 - 1135, determină CMMDC al celor N numere întregi conţinute în vectorul A. Schema logică a rutinei este prezentată în figura 5.5. Valoarea CMMDC determinată este atribuită variabilei R.C.

● Detrminarea CMMMC al maxim 20 de numere întregi

Aplicaţia este o generalizare a celei prezentate în 5.1.4. În realizarea sa s-au utilizat rutinele C3, R3, S2 (anexa A) şi R7 definită în cadrul acestei aplicaţii.

Rutina R7 permite determinarea CMMMC al celor N numere întregi depuse în vectorul A. Schema logică a rutinei este ilustrată în figura 5.6.

Aplicaţia este prezentată în listingul 5.6 împreună cu textul sursă al rutinei R7 (liniile 1180 - 1280).

```
10    PRINT "Determinarea C.M.M.M.C. al 2<=N<=20 numere intregi"
15    DIM A(20)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3060'Citeste N si A(N)
50     R.N=N : GOSUB 1180'C.M.M.M.C
60     IF ER% = 0 THEN CMMMC=R.C
70     GOSUB 5020'Afisare C.M.M.M.C.
80     INPUT "Continuati ? (Y/N) :",C$
90    WEND
100   STOP
1180   'C.M.M.M.C. al 2<=N<=20 numere intregi
1185   IF R.N<2 OR R.N>20 THEN ER%=-5 : GOTO 1275
1186   R.I=1
1187   WHILE A(R.I-1)<>0 AND R.I<=R.N :
           R.I=R.I+1 :
1189   WEND
1190   IF R.I <= R.N THEN R.MC=0 : GOTO 1275
1195   R.A=ABS(A(0)) : R.B=ABS(A(1)) : GOSUB 1065
1200   IF ER% <> 0 THEN GOTO 1275
1205   R.DCA=R.C
1210   R.MCA=ABS(A(0))/R.DCA*ABS(A(1))
1215   R.I=3 : R.MC=R.MCA
1220   WHILE R.I <= R.N AND ER% = 0
1225     R.A=R.DCA : R.B=ABS(A(R.I-1)) : GOSUB 1065
1230     IF ER% <> 0 THEN GOTO 1270
1235     R.MCB=R.DCA/R.C*ABS(A(R.I-1))
1240     R.DCA=R.C
1245     R.A=R.MCA : R.B=R.MCB : GOSUB 1065
1250     IF ER% <> 0 THEN GOTO 1270
1255     R.MC=R.MCA/R.C*R.MCB
1260     R.MCA=R.MC
1265     R.I=R.I+1
1270   WEND
1275   IF ER% = 0 THEN R.C=R.MC
1280   RETURN
MERGE "C3"
Ok
MERGE "R3"
```

```
Ok
MERGE "S2"
Ok
RUN
Determinarea C.M.M.M.C. al 2<=N<=20 numere intregi
Precizati N (2<=N<=20) : 4
A( 0 ) = 2
A( 1 ) = 3
A( 2 ) = 4
A( 3 ) = 5
C.M.M.D.C.= 60
Continuati ? (Y/N) :Y
Precizati N (2<=N<=20) : 5
A( 0 ) = 10
A( 1 ) = 21
A( 2 ) = 35
A( 3 ) = 3
A( 4 ) = 42
C.M.M.D.C.= 210
Continuati ? (Y/N) :N
Break in 100
Ok
```

Listing 5.6

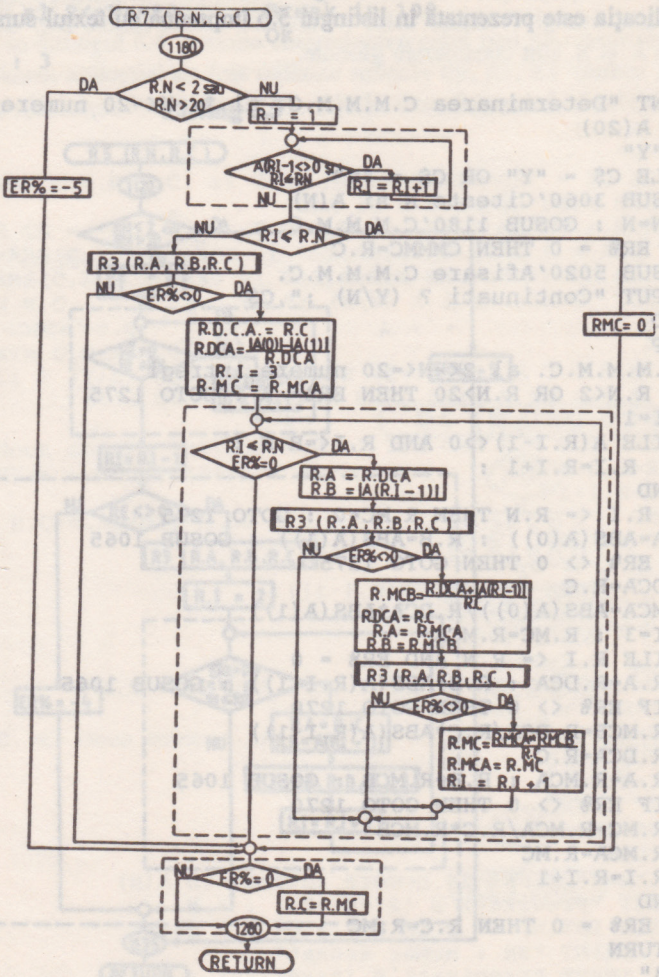


Fig. 5.6

Observaţie. Utilizarea acestei aplicaţii in determinarea CMMMC al mai multor numere întregi trebuie făcută cu atenţie deoarece, dacă se lucrează cu numere relativ mari sau cu mai multe numere prime între ele, în procesul de multiplicare se poate depăşi capacitatea maximă de reprezentare internă a numerelor întregi şi, ca atare, rezultatul va fi eronat.

● Generarea de numere prime

Aplicaţia permite determinarea numerelor prime mai mari sau egale cu 2 şi mai mici sau egale cu un număr `N` (maxim 1000).

În cadrul acestei aplicaţii este definită rutina `R8`, listingul 5.7, liniile 1300 - 1375, care, utilizînd algoritmul lui Euclid, determină şi depune în vectorul `PRIM` toate cele `NRPRIM` numere prime ce aparţin intervalului `[2, R.N]`. Rutina utilizează variabila de **intrare** `R.N` şi variabilele auxiliare `R.I, R.K, R.NF, R.N1` şi vectorul `R.AS` care, după determinarea numerelor prime aparţinînd intervalului dat, este şters (zona de memorie ocupată de acesta este disponibilizată). Schema logică a rutinei este prezentată în figura 5.7.

```
10    PRINT "Genereaza numerele prime <= N <= 1000"
15    DIM PRIM(168)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3040 : N=C.A'Citeste si valideaza N
50     R.N=N : GOSUB 1300'Generare
60     GOSUB 100'Afisare
70     INPUT "Continuati ? (Y/N) :",C$
80    WEND
90    STOP
100    '
105    IF ER% = 0
           THEN PRINT "Numerele prime <=";N;" sint :" :
       FOR I = 0 TO NRPRIM-1 STEP 12 :
       GOSUB 115 :
       NEXT :
           ELSE PRINT "ER = ";ER%
110    RETURN
115    '
120    IF I+10 > NRPRIM-1
           THEN J1=NRPRIM-1
           ELSE J1=I+10
125    FOR J = I TO J1 :
           PRINT USING "####";PRIM(J); :
       NEXT
130    IF I+11 <= NRPRIM-1
           THEN PRINT USING "####";PRIM(I+11)
           ELSE PRINT
135    RETURN
1300   'Determina numerele prime 2<=R.N<=1000
1305   IF R.N<2 OR R.N>1000 THEN ER%=-6 : GOTO 1375
1310   DIM R.AS(998)
1315   FOR R.I=2 TO R.N
1316     R.AS(R.I-2)=R.I
1317   NEXT
```

```
1320   R.NF=INT(SQR(R.N))
1325   R.K=2
1330   WHILE R.K <= R.NF
1335      IF R.AS(R.K-2) <> 0
       THEN R.N1=R.AS(R.K-2)*2 :
      WHILE R.N1 <= R.N :
       R.AS(R.N1-2)=0 :
       R.N1=R.N1+R.AS(R.K-2) :
      WEND
1336      R.K=R.K+1
1340   WEND
1345   R.K=0
1350   FOR R.I=0 TO R.N-2
1351      IF R.AS(R.I) <> 0
       THEN PRIM(R.K)=R.AS(R.I) :
      R.K=R.K+1
1355   NEXT
1360   NRPRIM=R.K
1365   ERASE R.AS
1370   ER%=0
1375   RETURN
MERGE "C2"
Ok
RUN
Genereaza numerele prime <= N <= 1000
N = 50
Numerele prime <= 50  sint :
   2   3   5   7  11  13  17  19  23  29  31  37
  41  43  47
Continuati ? (Y/N) :Y
N = 100
Numerele prime <= 100  sint :
   2   3   5   7  11  13  17  19  23  29  31  37
  41  43  47  53  59  61  67  71  73  79  83  89
  97
Continuati ? (Y/N) :Y
N = 1000
Numerele prime <= 1000  sint :
   2   3   5   7  11  13  17  19  23  29  31  37
  41  43  47  53  59  61  67  71  73  79  83  89
  97 101 103 107 109 113 127 131 137 139 149 151
 157 163 167 173 179 181 191 193 197 199 211 223
 227 229 233 239 241 251 257 263 269 271 277 281
 283 293 307 311 313 317 331 337 347 349 353 359
 367 373 379 383 389 397 401 409 419 421 431 433
 439 443 449 457 461 463 467 479 487 491 499 503
 509 521 523 541 547 557 563 569 571 577 587 593
 599 601 607 613 617 619 631 641 643 647 653 659
 661 673 677 683 691 701 709 719 727 733 739 743
 751 757 761 769 773 787 797 809 811 821 823 827
 829 839 853 857 859 863 877 881 883 887 907 911
 919 929 937 941 947 953 967 971 977 983 991 997
Continuati ? (Y/N) :N
Break in 90
Ok
```

Listing 5.7

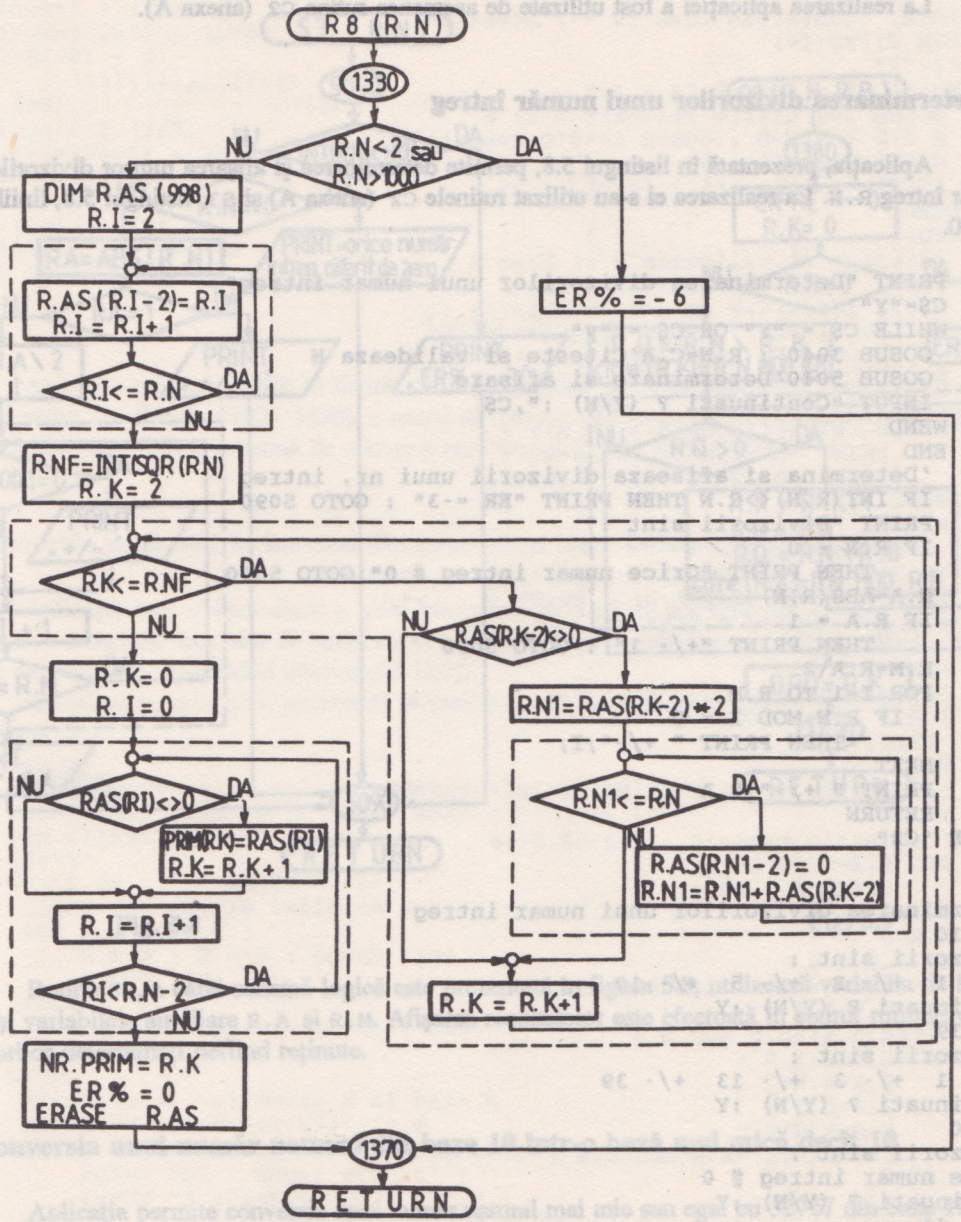

R 8 (R.N)
1330
R.N<2 sau R.N>1000
NU
DA
DIM R.AS (998)
R.I = 2
ER% = -6
R.AS(R.I-2)=R.I
R.I = R.I+1
R.I<=R.N
DA
NU
R.NF=INT(SQR(R.N)
R.K = 2
R.K<=R.NF
DA
NU
R.AS(R.K-2)<>0
NU
DA
R.K = 0
R.I = 0
R.N1=R.AS(R.K-2)*2
R.AS(R.I)<>0
NU
DA
R.N1<=R.N
DA
NU
PRIM(R.K)=RAS(RI)
R.K=R.K+1
R.AS(R.N1-2)=0
R.N1=R.N1+R.AS(R.K-2)
R.I = R.I+1
RI<R.N-2
DA
NU
R.K = R.K+1
NR.PRIM = R.K
ER% = 0
ERASE R.AS
1370
RETURN

Fig. 5.7

La realizarea aplicaţiei a fost utilizate de asemenea rutina C2 (anexa A).

● Determinarea divizorilor unui număr întreg

Aplicaţia, prezentată în listingul 5.8, permite determinarea şi afişarea tuturor divizorilor unui număr întreg R.N. La realizarea ei s-au utilizat rutinele C2 (anexa A) şi S3, listingul 5.8, liniile 5040 - 5090.

```
10   PRINT "Determinarea divizorilor unui numar intreg"
20   C$="Y"
30   WHILE C$ = "Y" OR C$ = "y"
40    GOSUB 3040 : R.N=C.A'Citeste si valideaza N
50    GOSUB 5040'Determinare si afisare
60    INPUT "Continuati ? (Y/N) :",C$
70   WEND
80   END
5040  'Determina si afiseaza divizorii unui nr. intreg
5045  IF INT(R.N)<>R.N THEN PRINT "ER =-3" : GOTO 5090
5050  PRINT "Divizorii sint :"
5055  IF R.N = 0
        THEN PRINT "Orice numar intreg # 0":GOTO 5090
5060  R.A=ABS(R.N)
5065  IF R.A = 1
        THEN PRINT "+/- 1" : GOTO 5090
5070  R.M=R.A\2
5075  FOR I=1 TO R.M
5076    IF R.N MOD I = 0
          THEN PRINT " +/-";I;
5080  NEXT
5085  PRINT " +/-";R.A
5090  RETURN
MERGE "C2"
Ok
RUN
Determinarea divizorilor unui numar intreg
N = 10
Divizorii sint :
 +/- 1  +/- 2  +/- 5  +/- 10
Continuati ? (Y/N) :Y
N = 39
Divizorii sint :
 +/- 1  +/- 3  +/- 13  +/- 39
Continuati ? (Y/N) :Y
N = 0
Divizorii sint :
Orice numar intreg # 0
Continuati ? (Y/N) :Y
N = -41
Divizorii sint :
 +/- 1  +/- 41
Continuati ? (Y/N) :N
Ok
```

Listing 5.8

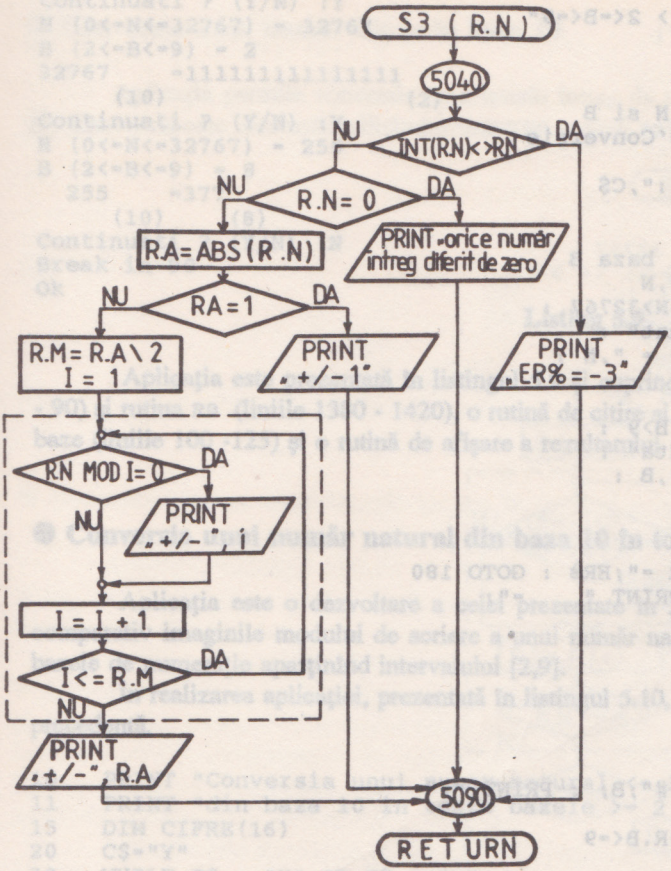


Fig. 5.8

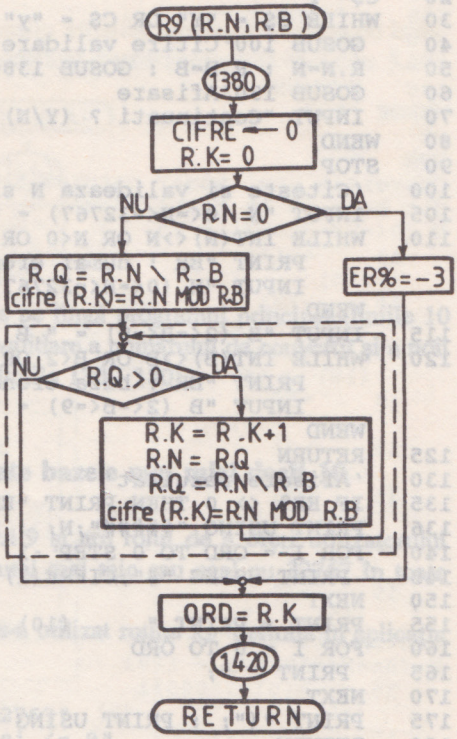


Fig. 5.9

Rutina S3, a cărei schemă logică este prezentată în figura 5.8, utilizează variabila de intrare R.N şi variabilele auxiliare R.A şi R.M. Afişarea rezultatelor este efectuată în cadrul rutinei, valorile divizorilor determinaţi nefiind reţinute.

● Conversia unui număr natural din baza 10 într-o bază mai mică decît 10

Aplicaţia permite conversia unui număr natural mai mic sau egal cu 32767 din baza 10 într-o bază aparţinînd intervalului [2,9]. Conversia este realizată cu rutina R9 a cărei schemă logică este prezentată în figura 5.9.

Rutina R9, listingul 5.9, liniile 1380 - 1420, utilizează variabilele de intrare R.N (numărul de convertit) şi R.B (noua bază), rezultatul (sub forma unui şir de cifre) fiind depus în vectorul CIFRE. Ordinul de mărime (numărul de cifre) al rezultatului este depus în variabila ORD.

```
10    PRINT "Conversie baza 10 --> 2<=B<=9"
15    DIM CIFRE(16)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 100'Citire validare N si B
50     R.N=N : R.B=B : GOSUB 1380'Conversie
60     GOSUB 130'Afisare
70     INPUT "Continuati ? (Y/N) :",C$
80    WEND
90    STOP
100    'Citeste si valideaza N si baza B
105    INPUT "N (0<=N<=32767) = ",N
110    WHILE INT(N)<>N OR N<0 OR N>32767 :
           PRINT "ER : numar eronat" :
           INPUT "N (0<=N<=32767) = ",N :
       WEND
115    INPUT "B (2<=B<=9) = ",B
120    WHILE INT(B)<>B OR B<2 OR B>9 :
           PRINT "ER : baza eronata" :
           INPUT "B (2<=B<=9) = ",B :
       WEND
125    RETURN
130    'Afisare rezultat
135    IF ER% <> 0 THEN PRINT "ER =";ER% : GOTO 180
136    PRINT USING "#####";N; : PRINT "     =";
140    FOR I = ORD TO 0 STEP -1
145     PRINT USING "#";CIFRE(I);
150    NEXT
155    PRINT : PRINT "      (10) ";
160    FOR I = 0 TO ORD
165     PRINT " ";
170    NEXT
175    PRINT "("; : PRINT USING "#";B; : PRINT ")"
180    RETURN
1380   'Conversie baza 10 --> 2<=R.B<=9
1385   FOR R.I=0 TO 15 :
           CIFRE(R.I)=0 :
       NEXT
1390   R.K=0
1395   IF R.N < 0
           THEN ER%=-3 : GOTO 1415
1400   R.Q=R.N\R.B
1405   CIFRE(R.K)=R.N MOD R.B
1410   WHILE R.Q > 0 :
           R.K=R.K+1 :
           R.N=R.Q :
           R.Q=R.N\R.B :
           CIFRE(R.K)=R.N MOD R.B :
       WEND
1415   ORD=R.K
1420   RETURN
RUN
Conversie baza 10 --> 2<=B<=9
N (0<=N<=32767) = 15
B (2<=B<=9) = 2
   15     =1111
     (10)      (2)
```

```
Continuati ? (Y/N) :Y
N (0<=N<=32767) = 32767
B (2<=B<=9) = 2
32767     =111111111111111
     (10)                (2)
Continuati ? (Y/N) :Y
N (0<=N<=32767) = 255
B (2<=B<=9) = 8
  255     =377
     (10)     (8)
Continuati ? (Y/N) :N
Break in 90
Ok
```

Listing 5.9

Aplicaţia este prezentată în listingul 5.9 şi cuprinde pe lîngă programul principal (liniile 10 - 90) şi rutina R9 (liniile 1380 - 1420), o rutină de citire şi validare a numărului de convertit şi a noii baze (liniile 100 -125) şi o rutină de afişare a rezultatului (liniile 130 - 180).

● Conversia unui număr natural din baza 10 în toate bazele mai mici decît 10

Aplicaţia este o dezvoltare a celei prezentate în 5.1.9 şi are rolul de a oferi utilizatorului comparativ imaginile modului de scriere a unui număr natural mai mic sau egal cu 32767 în toate bazele de numeraţie aparţinînd intervalului [2,9].

În realizarea aplicaţiei, prezentată în listingul 5.10, s-a utilizat rutina R9 definită în aplicaţia precedentă.

```
10    PRINT "Conversia unui numar natural <= 32767"
11    PRINT "din baza 10 in toate bazele >= 2 si <= 9"
15    DIM CIFRE(16)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 100'Citire validare N si B
50     FOR B = 2 TO 9 :
           R.N=N : R.B=B : GOSUB 1380 :
           GOSUB 130
60     NEXT
70     INPUT "Continuati ? (Y/N) :",C$
80    WEND
90    STOP
100    'Citeste si valideaza N si baza B
105    INPUT "N (0<=N<=32767) = ",N
110    WHILE INT(N)<>N OR N<0 OR N>32767 :
           PRINT "ER : numar eronat" :
           INPUT "N (0<=N<=32767) = ",N :
       WEND
125    RETURN
130    'Afisare rezultat
135    IF ER% <> 0 THEN PRINT "ER =";ER% : GOTO 180
136    PRINT USING "######";N; : PRINT "     =";
140    FOR I = ORD TO 0 STEP -1
145     PRINT USING "#";CIFRE(I);
```

```
150   NEXT
155   PRINT : PRINT "      (10) ";
160   FOR I = 0 TO ORD
165    PRINT " ";
170   NEXT
175   PRINT "("; : PRINT USING "#";B; : PRINT ")"
180   RETURN
MERGE "R9"
Ok
RUN
Conversia unui numar natural <= 32767
din baza 10 in toate bazele >= 2 si <= 9
N (0<=N<=32767) = 15
   15     =1111
     (10)      (2)
   15     =120
     (10)      (3)
   15     =33
     (10)      (4)
   15     =30
     (10)      (5)
   15     =23
     (10)      (6)
   15     =21
     (10)      (7)
   15     =17
     (10)      (8)
   15     =16
     (10)      (9)
Continuati ? (Y/N) :Y
N (0<=N<=32767) = 1283
   15     =10100000011
     (10)      (2)
   15     =1202112
     (10)      (3)
   15     =110003
     (10)      (4)
   15     =20113
     (10)      (5)
   15     =5535
     (10)      (6)
   15     =3512
     (10)      (7)
   15     =2403
     (10)      (8)
   15     =1675
     (10)      (9)
Continuati ? (Y/N) :N
Break in 90
Ok
```

Listing 5.10

● Conversia unui număr binar în baza 10

Aplicaţia permite conversia unui număr întreg de maxim 16 cifre binare din baza 2 în baza 10. Conversia este efectuată utilizînd rutina `R10` a cărei schemă logică este prezentată în figura 5.10.

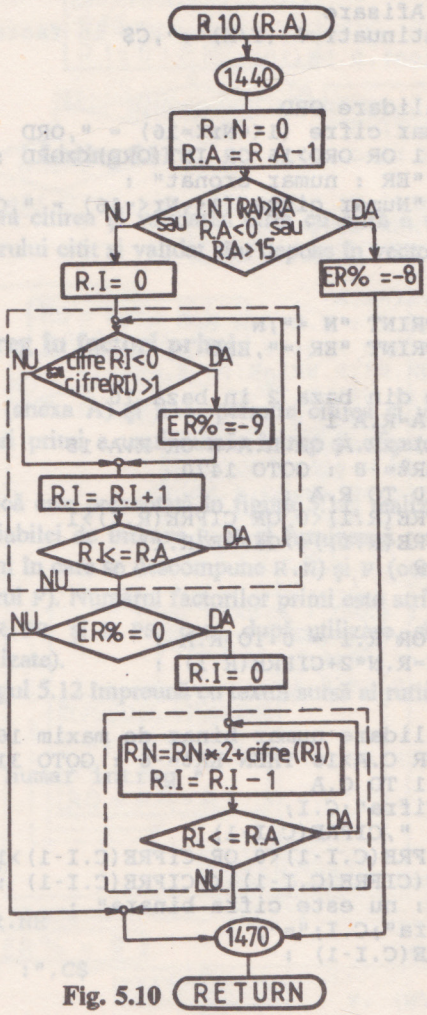


Fig. 5.10

Rutina `R10` asigură conversia numărului ale cărui `ORD` cifre binare sînt depuse în vectorul `CIFRE`. Numărul întreg în baza 10 rezultat este atribuit variabilei `R.A`.

Aplicaţia este prezentată în listingul 5.11 şi conţine pe lîngă rutina `R10` (liniile 1440 - 1470) şi textul sursă al rutinei `C4` (liniile 3100 - 3130).

```
10    PRINT "Conversia in baza 10 a unui numar binar"
15    DIM CIFRE(16)
```

```
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 100 'Citire validare N
50     C.A=ORD : GOSUB 3100'Citire validare CIFRE
60     IF ER% = 0
          THEN R.A=ORD : GOSUB 1440 : N=R.N
70     GOSUB 135'Afisare
80     INPUT "Continuati ? (Y/N) :",C$
90    WEND
95    STOP
100    'Citire validare ORD
105    INPUT "Numar cifre (1<=Nr<=16) = ",ORD
110    WHILE ORD<1 OR ORD>16 OR INT(ORD)<>ORD :
          PRINT "ER : numar eronat" :
          INPUT "Numar cifre (1<=Nr<=16) = ",ORD :
       WEND
115    RETURN
135    'Afisare
140    IF ER% = 0
          THEN PRINT "N =";N
          ELSE PRINT "ER =",ER%
145    RETURN
1440   'Conversie din baza 2 in baza 10
1445   R.N=0 : R.A=R.A-1
1450   IF INT(R.A)<>R.A OR R.A<0 OR R.A>15
          THEN ER%=-8 : GOTO 1470
1455   FOR R.I = 0 TO R.A :
          IF CIFRE(R.I)<0 OR CIFRE(R.I)>1
       OR INT(CIFRE(R.I))<>CIFRE(R.I)
       THEN ER%=-9
1460   NEXT
1465   IF ER% = 0
          THEN FOR R.I = 0 TO R.A :
             R.N=R.N*2+CIFRE(R.I) :
           NEXT
1470   RETURN
3100   'Citire validare numar binar de maxim 16 cifre
3105   IF C.A<1 OR C.A>16 THEN ER%=-6 : GOTO 3130
3110   FOR C.I = 1 TO C.A
3111     PRINT "Cifra";C.I;
3112     INPUT "= ",CIFRE(C.I-1)
3113     WHILE CIFRE(C.I-1)<0 OR CIFRE(C.I-1)>1
           OR INT(CIFRE(C.I-1))<>CIFRE(C.I-1) :
       PRINT "ER : nu este cifra binara" :
       PRINT "Cifra";C.I;"="; :
       INPUT CIFRE(C.I-1) :
       WEND
3120   NEXT
3130   RETURN
RUN
Conversia in baza 10 a unui numar binar
Numar cifre (1<=Nr<=16) = 4
Cifra 1 = 1
Cifra 2 = 1
Cifra 3 = 1
Cifra 4 = 1
N = 15
```

```
Continuati ? (Y/N) :Y
Numar cifre (1<=Nr<=16) = 6
Cifra 1 = 1
Cifra 2 = 0
Cifra 3 = 1
Cifra 4 = 0
Cifra 5 = 1
Cifra 6 = 0
N = 42
Continuati ? (Y/N) :N
Break in 95
Ok
```

Listing 5.11

Rutina C4 are rolul de a asigura citirea şi validarea cifră cu cifră a unui număr binar de maxim 16 cifre. Cifrele binare ale numărului citit şi validat sînt depuse în vectorul CIFRE.

● Descompunerea unui număr întreg în factori primi

Aplicaţia, utilizînd rutinele C2 (anexa A) şi R11, permite citirea şi validarea unui număr întreg, descompunerea acestuia în factori primi a unui număr întreg şi afişarea ordonată şi într-o formă intuitivă a acestora.

Rutina R11, a cărui schemă logică este prezentată în figura 5.11, realizează descompunerea în factori primi a numărului atribuit variabilei de **intrare** R.N şi furnizează rezultatul în vectorii F (conţinînd lista ordonată a factorilor primi în care se descompune R.N) şi P (conţinînd lista puterilor factorilor primi corespunzători din vectorul F). Numărul factorilor primi este atribuit variabilei R.NR. Rutina utilizează vectorii de manevră R.FS şi R.PS care, după utilizare, sînt şterşi (zonele de memorie afectate acestora sînt disponibilizate).

Aplicaţia este prezentată în listingul 5.12 împreună cu textul sursă al rutinei R11 (liniile 1480 - 1565).

```
10    PRINT "Descompunerea unui numar intreg "
11    PRINT "in factori primi"
15    DIM F(100),P(100)
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3040 : N=C.A
50     R.N=N : GOSUB 1480 : NR=R.NR
60     GOSUB 100
70     INPUT "Continuati ? (Y/N) :",C$
80    WEND
90    STOP
100    'Afisare
105    IF ER% = 0
           THEN GOSUB 115
           ELSE PRINT "ER = ";ER%
110   RETURN
115
120   PRINT "N =";
125   IF N=0
```

```
          THEN PRINT NR
          ELSE FOR I=0 TO NR-1:
            PRINT F(I);"^";P(I);"*";:
       NEXT:
       PRINT F(NR);"^";P(I)
130    PRINT
135    RETURN
1480   'Descompune un numar intreg R.N in factori primi
1485   DIM R.FS(100),R.PS(100)
1490   R.NR=0
1495   FOR R.I = 0 TO 100 :
          R.FS(R.I)=0 :
          R.PS(R.I)=0
1500   NEXT
1505   IF R.N<>INT(R.N) THEN ER%=-3 : GOTO 1560
1510   IF R.N=0 THEN GOTO 1555
1515   R.N=ABS(R.N)
1520   R.I=2
1525     WHILE R.I<=R.N
1530     R.Q=R.N\R.I : R.R=R.N MOD R.I
1535     IF R.R=0 AND R.Q<>0
            THENR.PS(R.NR)=R.PS(R.NR)+1:
            R.FS(R.NR)=R.I:
            R.R=R.Q MOD R.I:
            R.N=R.Q:R.Q=R.N\R.I:
            WHILER.R=0 AND R.Q<>0:
            R.PS(R.NR)=R.PS(R.NR)+1:
            R.R=R.Q MOD R.I:
            R.N=R.Q:R.Q=R.N\R.I:
       WEND:
       R.NR=R.NR+1
1540   R.I=R.I+1 : WEND : R.NR=R.NR-1
1545   FOR R.I = 0 TO R.NR :
          F(R.I)=R.FS(R.I) :
          P(R.I)=R.PS(R.I)
1550   NEXT
1555   ER%=0
1560   ERASE R.FS,R.PS
1565   RETURN
MERGE "C2"
Ok
RUN
Descompunerea unui numar intreg
in factori primi
N = 36
N = 2 ^ 2 * 3 ^ 2

Continuati ? (Y/N) :Y
N = 256
N = 2 ^ 8

Continuati ? (Y/N) :N
Break in 90
Ok
```

Listing 5.12

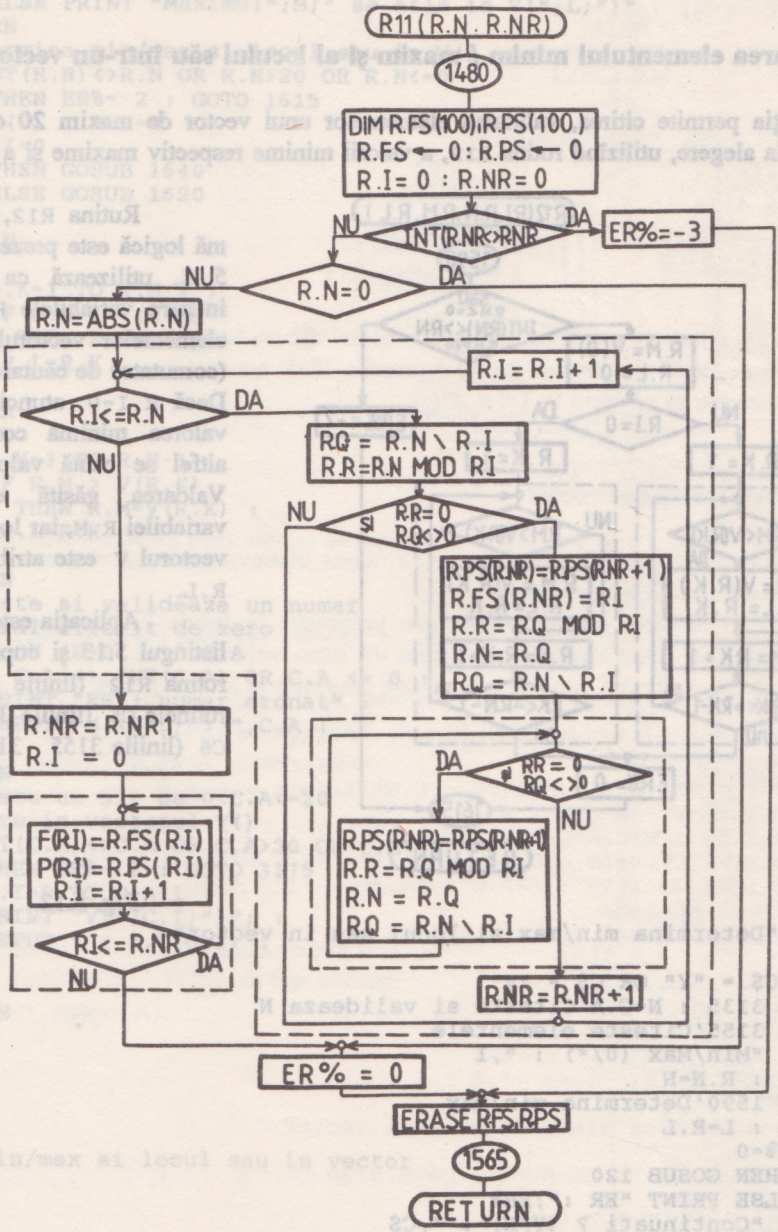

R11(R.N, R.NR)
1480
DIM R.FS(100),R.PS(100)
R.FS ← 0 : R.PS ← 0
R.I = 0 : R.NR = 0
INT(R.NR<>R.NR
DA
ER%=-3
NU
R.N = 0
DA
NU
R.N=ABS(R.N)
R.I = R.I + 1
R.I<=R.N
DA
NU
R.Q = R.N \ R.I
R.R=R.N MOD R.I
NU
SI
R.R=0
R.Q<>0
DA
R.PS(R.NR)=R.PS(R.NR+1)
R.FS(R.NR) = R.I
R.R = R.Q MOD R.I
R.N = R.Q
R.Q = R.N \ R.I
R.NR = R.NR1
R.I = 0
DA
SI
R.R = 0
R.Q<>0
NU
F(R.I)=R.FS(R.I)
P(R.I)=R.PS(R.I)
R.I = R.I+1
R.PS(R.NR)=R.PS(R.NR+1)
R.R = R.Q MOD R.I
R.N = R.Q
R.Q = R.N \ R.I
RI<=R.NR
DA
NU
R.NR = R.NR+1
ER% = 0
ERASE R.FS,R.PS
1565
RETURN

Fig. 5.11

5.2 Vectori

● Determinarea elementului minim / maxim şi al locului său într-un vector

Aplicaţia permite citirea, validarea elementelor unui vector de maxim 20 componente şi determinarea, la alegere, utilizînd rutina R12, a valorii minime respectiv maxime şi a locului său în vectorul dat.

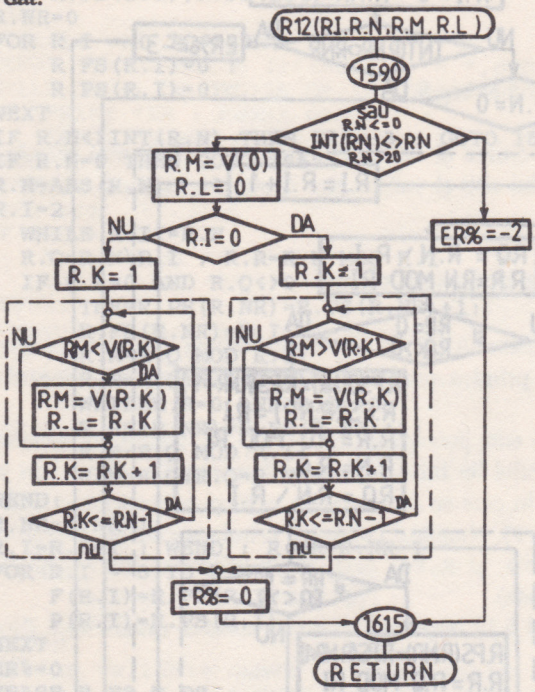


Fig. 5.12

Rutina R12, a cărei schemă logică este prezentată în figura 5.12, utilizează ca variabile de **intrare** variabilele R.N (numărul elementelor vectorului V) şi R.I (comutator de căutare).
Dacă R.I=0 atunci se determină valorea minimă conţinută în V, altfel se caută valoarea maximă. Valoarea găsită este atribuită variabilei R.M, iar locul acestuia în vectorul V este atribuită variabilei R.L.

Aplicaţia este prezentată în listingul 5.13 şi cuprinde pe lîngă rutina R12 (liniile 1590 - 1655), rutinele C5 (liniile 3135 - 3150) şi C6 (liniile 3155 - 3175).

```
10    PRINT "Determina min/max si locul sau in vector"
20    C$="Y"
30    WHILE C$ = "Y" OR C$ = "y"
40     GOSUB 3135 : N=C.A'Citeste si valideaza N
50     GOSUB 3155'Citeste elementele
60     INPUT "Min/Max (0/*) : ",I
70     R.I=I : R.N=N
80     GOSUB 1590'Determina min/max
90     M=R.M : L=R.L
100    IF ER%=0
           THEN GOSUB 120
           ELSE PRINT "ER :";ER%
105    INPUT "Continuati ? (Y/N) : ",C$
110   WEND
115   END
120    'Afisare
125    IF I=0
```

```
        THEN PRINT "Minimul";M;" se afla in V(";L;")"
        ELSE PRINT "Maximul";M;" se afla in V(";L;")"
130   RETURN
1590  'Determina min/max si locul sau in V()
1595  IF INT(R.N)<>R.N OR R.N>20 OR R.N<=0
        THEN ER%=-2 : GOTO 1615
1600  R.M=V(0) : R.L=0
1605  IF R.I=0
        THEN GOSUB 1640
        ELSE GOSUB 1620
1610  ER%=0
1615  RETURN
1620  'Max
1625  FOR R.K=1 TO R.N-1 :
        IF R.M < V(R.K)
          THEN R.M=V(R.K) :
        R.L=R.K
1630  NEXT
1635  RETURN
1640  'Min
1645  FOR R.K=1 TO R.N-1 :
        IF R.M > V(R.K)
          THEN R.M=V(R.K) :
        R.L=R.K
1650  NEXT
1655  RETURN
3135  'Citeste si valideaza un numar
3136  'natural diferit de zero
3140  INPUT "N (N>0) : ",C.A
3145  WHILE C.A <> INT(C.A) OR C.A <= 0 :
        PRINT "ER : numar eronat" :
        INPUT "N (N>0) : ",C.A :
      WEND
3150  RETURN
3155  'Citeste un sir de 0<C.A<=20
3156  'numere in vectorul V()
3160  IF INT(C.A)<>C.A OR C.A>20 OR C.A<=0
        THEN ER%=-2 : GOTO 3175
3165  FOR C.I=0 TO C.A-1 :
        PRINT "V(";C.I;")"; :
        INPUT " = ",V(C.I) :
      NEXT
3170  ER%=0
3175  RETURN
MERGE "R10"
Ok
MERGE "C5"
Ok
RUN
Determina min/max si locul sau in vector
N (N>0) : 5
V( 0 ) = 8
V( 1 ) = 0
V( 2 ) = 7
V( 3 ) = 2
V( 4 ) = 1
Min/Max (0/*) : 0
```

```
Minimul 0 se afla in V( 1 )
Continuati ? (Y/N) : Y
N (N>0) : 5
V( 0 ) = 8
V( 1 ) = 0
V( 2 ) = 7
V( 3 ) = 2
V( 4 ) = 1
Min/Max (0/*) : 1
Maximul 8 se afla in V( 0 )
Continuati ? (Y/N) : N
Ok
```

Listing 5.13

Rutina C5 citeşte şi validează un număr natural diferit de 0.

Rutina C6 citeşte şi validează un şir de maxim 20 de numere întregi. Cele C.A numere citite şi validate sînt depuse în vectorul V.

● Suma elementelor unui vector

Aplicaţia determină utilizînd rutina R13 suma elementelor unui vector V de maxim 20 componente, precum şi afişarea acestuia.

Rutina R13, a cărei schemă logică este prezentată în figura 5.13, utilizează variabila de **intrare** R.N (numărul componentelor vectorului V), suma valorilor elementelor vectorului V fiind atribuită variabilei R.S. Textul sursă al rutinei este prezentat în listingul 5.14, liniile 1660 - 1675.

```
10    PRINT "Insumarea elementelor unui vector"
20    DIM V(20)
30    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
40    WHILE C$ = "Y" OR C$ = "y"
50     GOSUB 3180 : N=C.A'Citeste N
60     GOSUB 3155'Citeste elementele
70     R.N=N : GOSUB 1660 : S=R.S'Suma
80     GOSUB 120'Afisare
90     INPUT "Continuati ? (Y/N) : ",C$
100   WEND
110   END
120    'Afisare
130    IF ER%<>0
          THEN PRINT "ER =";ER%
          ELSE PRINT "Suma =";S
140    RETURN
1660   'Determina suma elementelor unui vector V()
1665   R.S=0
1670   IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
          THEN ER%=-2
          ELSE FOR R.I=0 TO R.N-1 :
           R.S=R.S+V(R.I) :
       NEXT :
         ER%=0
1675   RETURN
MERGE "C6"
```

```
Ok
MERGE "C7"
Ok
RUN
Insumarea elementelor unui vector
N( 2 <= N <= 20 ) = 4
V( 0 ) = 7
V( 1 ) = -3
V( 2 ) = 6
V( 3 ) = 5
Suma = 15
Continuati ? (Y/N) : N
Ok
```

Listing 5.14

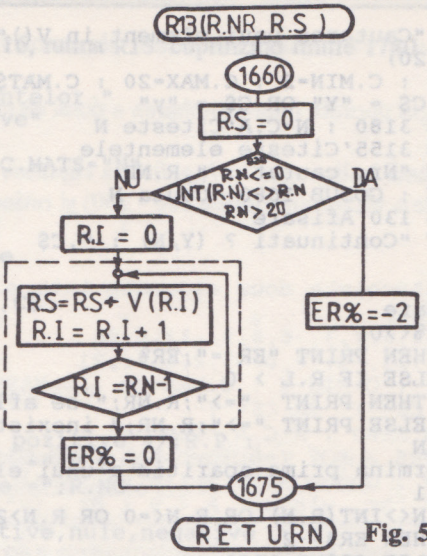


Fig. 5.13

În realizarea aplicației s-au utilizat de asemenea rutinele C6 și C7 (anexa A).

● Căutarea unei element într-un vector

Aplicația, prezentată în listingul 5.15, permite căutarea în vectorul V a unui element (număr întreg) oarecare și, în cazul în care acesta există, afișează locul său în V. Dacă elementul nu aparține vectorului V se afișează un mesaj corespunzător.

Căutarea elementului dorit R.NR în vectorul V este realizată de rutina R14, a cărei schemă logică este prezentată în figura 5.14. Dacă R.NR aparține vectorului V atunci R.L conține poziția acestuia în V, altfel R.L=-1.

În realizarea aplicației în afara rutinei R14 (listingul 5.15, liniile 1680 - 1730), au fost utilizate rutinele C6 și C7 (anexa A).

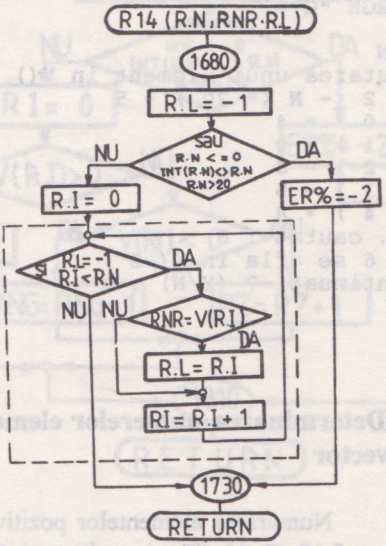


Fig. 5.14

```
10    PRINT "Cautarea unui element in V()"
20    DIM V(20)
30    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
40    WHILE C$ = "Y" OR C$ = "y"
50     GOSUB 3180 : N=C.A'Citeste N
60     GOSUB 3155'Citeste elementele
70     INPUT "Nr. cautat : ",R.NR
80     R.N=N : GOSUB 1680 'Cauta N
90     GOSUB 130'Afisare
100    INPUT "Continuati ? (Y/N) : ",C$
110   WEND
120   END
130    'Afisare
140    IF ER%<>0
           THEN PRINT "ER =";ER%
           ELSE IF R.L > 0
            THEN PRINT  "=>";R.NR;" se afla in V(";R.L;")"
            ELSE PRINT "=>";R.NR;" inexistent in V()
150   RETURN
1680  'Determina prima aparitie a unui element in V()
1685  R.L=-1
1690  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
          THEN ER%=-2
          ELSE GOSUB 1700
1695  RETURN
1700  '
1705  R.I=0
1710  WHILE R.L=-1 AND R.I<R.N
1715    IF R.NR=V(R.I) THEN R.L=R.I
1720    R.I=R.I+1
1725  WEND
1730  RETURN
MERGE "C6"
Ok
MERGE "C7"
Ok
RUN
Cautarea unui element in V()
N( 2 <= N <= 20 ) = 5
V( 0 ) = 4
V( 1 ) = 0
V( 2 ) = 6
V( 3 ) = 5
V( 4 ) = 8
Nr. cautat : 6
=> 6 se afla in V( 2 )
Continuati ? (Y/N) : N
Ok
```

Listing 5.15

● Determinarea numerelor elementelor pozitive, nule şi negative conţinute într-un vector

Numărarea elementelor pozitive, nule şi negative se realizează cu rutina `R15` prezentată în figura 5.15. După revenirea din rutină numerele elementelor pozitive, negative şi nule sînt date de valorile variabilelor `R.P`, `R.Z`, respectiv `R.NG`.

Aplicaţia este prezentată în listingul 5.16, rutina R15 cuprinzînd liniile 1740 - 1810.

```
10    PRINT "Determina numarul elementelor "
20    PRINT "pozitive,nule si negative"
30    DIM V(20)
40    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50    WHILE C$ = "Y" OR C$ = "y"
60     GOSUB 3180 : N=C.A'Citeste N
70     GOSUB 3155'Citeste elementele
80     R.N=N : GOSUB 1740'Determinare
90     GOSUB 130'Afisare
100    INPUT "Continuati ? (Y/N) : ",C$
110   WEND
120   END
130    'Afisare
140    IF ER%<>0
            THEN PRINT "ER =";ER%
            ELSE PRINT "Nr. elemente pozitive =";R.P :
         PRINT "Nr. elemente nule      =";R.Z :
         PRINT "Nr. elemente negative =";R.NG
150   RETURN
1740  'Detrminare nr. elemente pozitive,nule,negative
1745  R.P=0 : R.Z=0 : R.NG=0
1780  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
            THEN ER%=-2
            ELSE GOSUB 1790
1785  RETURN
1790  '
1795  FOR R.I=0 TO R.N-1
1800    IF V(R.I) > 0
            THEN R.P=R.P+1
            ELSE IF V(R.I) < 0
                    THEN R.NG=R.NG+1
                    ELSE R.Z=R.Z+1
1805  NEXT
1810  RETURN
MERGE "C6"
Ok
MERGE "C7"
Ok
RUN
Determina numarul elementelor
pozitive,nule si negative
N( 2 <= N <= 20 ) = 7
V( 0 ) = 4
V( 1 ) = 0
V( 2 ) = 6
V( 3 ) = -8
V( 4 ) = 8
V( 5 ) = -1
V( 6 ) = 5
Nr. elemente pozitive = 4
Nr. elemente nule     = 1
Nr. elemente negative = 2
Continuati ? (Y/N) : N
Ok
```

Listing 5.16

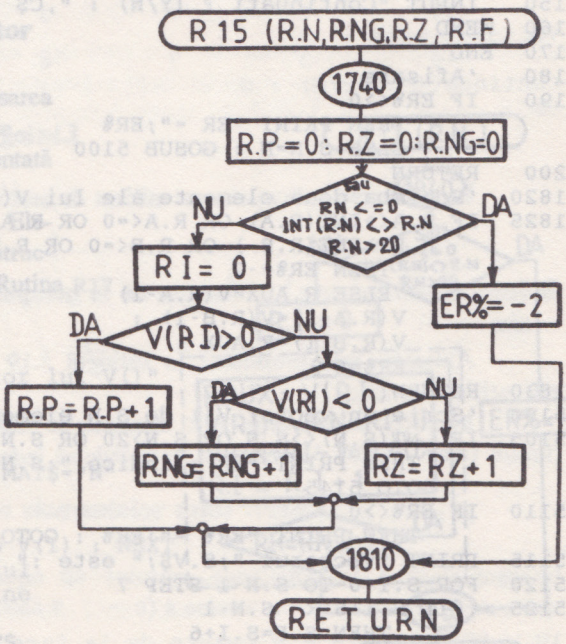


Fig. 5.15

● Interschimbarea a două elemente ale unui vector

Aplicaţia permite interschimbarea a două elemente ale vectorului V dat precizate prin poziţia lor în acesta.

Prezentată în listingul 5.17, aplicaţia cuprinde pe lîngă programul principal (liniile 10 - 170), rutina de afişare a vectorului (liniile 180 - 200) şi rutinele R16 (liniile 1820 - 1830) şi S4 (liniile 5100 - 5145).

```
10   PRINT "Schimba doua elemente inte ele"
20   DIM V(20)
30   DATA 10,6,7,3,-6,8,9,-11,6,24
35   DATA 4,11,-3,-8,0,-9,2,1,-1,21
40   C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50   WHILE C$="Y" OR C$="Y"
60    GOSUB 3180 : N=C.A'Alege N
70    RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80    S.N=N : GOSUB 5100'Scrie vectorul
90    PRINT "Nr. element A ( 0 < Nr. <=";N;")";
100   INPUT " = ",R.A
110   PRINT "Nr. element B ( 0 < Nr. <=";N;")";
120   INPUT " = ",R.B
130   R.N=N : GOSUB 1820'Schimba elemente
140   GOSUB 180'Afisare
150   INPUT "Continuati ? (Y/N) : ",C$
160  WEND
170  END
180   'Afisare
190   IF ER%<>0
         THEN PRINT "ER =";ER%
         ELSE S.N=N : GOSUB 5100
200   RETURN
1820  'Schimba doua elemente ale lui V() intre ele
1825  IF R.A<>INT(R.A) OR R.A<=0 OR R.A>R.N OR
      R.B<>INT(R.B ) OR R.B<=0 OR R.B>R.N
         THEN ER%=-2
         ELSE R.AUX=V(R.A-1) :
        V(R.A-1)=V(R.B-1) :
        V(R.B-1)=R.AUX :
        ER%=0
1830  RETURN
5100  'Scrie un vector V() de S.N elemente
5105  IF  NT(S.N)<>S.N OR S.N>20 OR S.N<=0
       THEN PRINT "ER : indice ";S.N;" eronat " :
      GOTO 5145
5110  IF ER%<>0
       THEN PRINT "ER% =";ER% : GOTO 5145
5115  PRINT "Vectorul ";S.V$;" este :"
5120  FOR S.I=0 TO S.N-1 STEP 7
5125    IF S.I+6 <= S.N-1
         THEN S.F=S.I+6
         ELSE S.F=S.N-1
5130     FOR S.K=S.I TO S.F :
      PRINT USING " ########";V(S.K); :
        NEXT
5135    PRINT
```

```
5140  NEXT
5145  RETURN
MERGE "C7"
Ok
RUN
Schimba doua elemente intre ele
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10          6          7          3          6          8
Nr. element A ( 0 < Nr. <= 6 ) = 3
Nr. element B ( 0 < Nr. <= 6 ) = 5
Vectorul este :
      10          6         -6          3          7          8
Continuati ? (Y/N) : N
Ok
```

Listing 5.17

Rutina `R16` interschimbă în vectorul `V` elementele de pe poziţiile `R.A` şi `R.B`. Dacă `R.A` sau `R.B` nu aparţin intervalului `[0, R.N]` se semnalează eroare (`ER%=-2`).

Rutina `S4` permite afişarea cu format a conţinutului vectorului `V`. De asemenea, prin variabila şir de caractere `S.V$` poate fi transmis şi inserat un comentariu înaintea afişării propriu-zise a valorilor componentelor vectorului `V`.

● Inversarea ordinii elementelor unui vector

Aplicaţia, prin rutina `R17`, permite inversarea ordinii elementelor vectorului `V` de `R.N` elemente.

Schema logică a rutinei `R17` este prezentată în figura 5.16.

Aplicaţia este prezentată în listingul 5.18. Elementele vectorului `V` sînt citite în linia 70 cu instrucţiunea `READ` din listele `DATA` (liniile 30 si 35). Rutina `R17` conţine liniile 1840 - 1850.

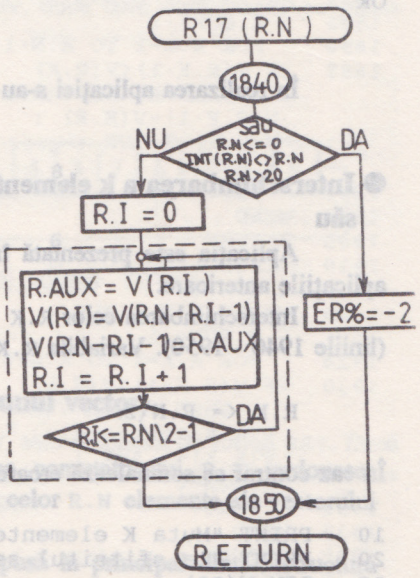


Fig. 5.16

```
10    PRINT "Schimba ordinea elementelor lui V()"
20    DIM V(20)
30    DATA 10,6,7,3,-6,8,9,-11,6,24
35    DATA 4,11,-3,-8,0,-9,2,1,-1,21
40    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50    WHILE C$="Y" OR C$="Y"
60     GOSUB 3180 : N=C.A'Alege N
70     RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80     S.N=N : GOSUB 5100'Scrie vectorul
90     R.N=N : GOSUB 1840'Schimba ordine
100    GOSUB 140'Afisare
110    INPUT "Continuati ? (Y/N) : ",C$
120   WEND
130   END
140    'Afisare
150    IF ER%<>0
```

```
          THEN PRINT "ER =";ER%
          ELSE S.N=N : GOSUB 5100
160    RETURN
1840   'Schimba ordinea elementelor lui V()
1845   IF R.N<>INT(R.N) OR R.N<=0 OR R.N>20
         THEN ER%=-2
         ELSE FOR R.I=0 TO R.N\2-1 :
          R.AUX=V(R.I) :
          V(R.I)=V(R.N-R.I-1) :
          V(R.N-R.I-1)=R.AUX :
       NEXT
1850   RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Schimba ordinea elementelor lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10        6         7         3        -6         8
Vectorul este :
       8        -6         3         7         6        10
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 5
Vectorul este :
      10        6         7         3        -6
Vectorul este :
      -6        3         7         6        10
Continuati ? (Y/N) : N
Ok
```

Listing 5.18

În realizarea aplicaţiei s-au utilizat de asemenea rutinele `C7` şi `S4` (anexa A).

● Interschimbarea a k elemente de la începutul unui vector cu k elemente de la sfîrşitul său

Aplicaţia este prezentată în listingul 5.19 şi constituie o dezvoltare a celor prezentate în aplicaţiile anterioare.

Interschimbarea celor `R.K` elemente se realizează cu rutina `R19` prezentată în listingul 5.19 (liniile 1940 - 1950). Variabila `R.K` trebuie să îndeplinească condiţia

```
R.K <= R.N\2.
```

În caz contrar se semnalează eroare.

```
10    PRINT "Muta K elemente de la inceputul lui V()"
20    PRINT "la sfirsitul sau si invers"
30    DIM V(20)
40    DATA 10,6,7,3,-6,8,9,-11,6,24
35    DATA 4,11,-3,-8,0,-9,2,1,-1,21
```

```
50    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
60    WHILE C$="Y" OR C$="Y"
70     GOSUB 3180 : N=C.A'Alege N
80     RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
90     S.N=N : GOSUB 5100'Scrie vectorul
100    PRINT "K ( 0 < K <=";N\2;")";
110   INPUT " = ",R.K
120    R.N=N : GOSUB 1940'Schimba
130    GOSUB 170'Afisare
140    INPUT "Continuati ? (Y/N) : ",C$
150   WEND
160   END
170    'Afisare
180    IF ER%<>0
            THEN PRINT "ER =";ER%
            ELSE S.N=N : GOSUB 5100
190    RETURN
1940   'Muta K elemente de la inceput la
1941   'sfirsitul lui V() si invers
1945   IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
          THEN ER%=-5
          ELSE IF R.K<>INT(R.K) OR R.K<=0 OR R.K>R.N\2
           THEN ER%=-2
           ELSE FOR R.I=0 TO R.K-1:
       R.AUX=V(R.I):
       V(R.I)=V(R.N-R.K+R.I):
       V(R.N-R.K+R.I)=R.AUX :
       NEXT:ER%=0
1950   RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Muta K elemente de la inceputul lui V()
la sfirsitul sau si invers
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10          6          7          3         -6          8
K ( 0 < K <= 3 ) = 2
Vectorul este :
      -6          8          7          3         10          6
Continuati ? (Y/N) : N
Ok
```

Listing 5.19

● Ordonarea crescătoare / descrescătoare a elementelor unui vector

Aplicaţia permite prin rutina `R18`, funcţie de valoarea comutatorului `R.I`, ordonarea crescătoare (dacă `R.I <> 0`) sau descrescătoare (dacă `R.I = 0`) a celor `R.N` elemente ale vectorului V.

Rutina `R18`, listingul 5.20, liniile 1860 - 1936, este compusă în principal dintr-o structură `IF_THEN_ELSE` care, funcţie de valoarea comutatorului `R.I` dirijează execuţia către rutina `Descrescator` (`R.I = 0`) sau `Crescator` (`R.I <> 0`). Ramurile fiind asemănătoare, în figura 5.17 se prezintă schema logică numai a rutinei `Crescator` (liniile 1910 - 1936).

```
10    PRINT "Ordonarea crescatoare descrescatoare a lui V()"
20    DIM V(20)
30    DATA 10,6,7,3,-6,8,9,-11,6,24
35    DATA 4,11,-3,-8,0,-9,2,1,-1,21
40    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50    WHILE C$="Y" OR C$="Y"
60     GOSUB 3180 : N=C.A'Alege N
70     RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80     S.N=N : GOSUB 5100'Scrie vectorul
90     INPUT "Crescator/Descrescator (*/0) : ",R.I
100    R.N=N : GOSUB 1860'Ordonare
170    GOSUB 300'Afisare
180    INPUT "Continuati ? (Y/N) : ",C$
190   WEND
200   END
300    'Afisare
310   IF ER%<>0
          THEN PRINT "ER =";ER%
          ELSE S.N=N : GOSUB 5100
320    RETURN
1860  'Ordonare crescatoare/descrescatoare
1861  IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
          THEN ER%=-5
          ELSE IF R.I=0
                 THEN GOSUB 1870 : ER%=0
                 ELSE GOSUB 1910 : ER%=0
1865   RETURN
1870   'Descrescator
1875   R.BUN=0
1880   WHILE NOT R.BUN
1885     R.BUN=-1
1890     FOR R.K=1 TO R.N-1
1895   IF V(R.K-1)<V(R.K)
           THEN R.AUX=V(R.K-1) :
        V(R.K-1)=V(R.K) :
        V(R.K)=R.AUX :
        R.BUN=0
1896     NEXT
1900   WEND
1905   RETURN
1910   'Crescator
1915   R.BUN=0
1920   WHILE NOT R.BUN
1921     R.BUN=-1
1925     FOR R.K=1 TO R.N-1
1930   IF V(R.K-1)>V(R.K)
           THEN R.AUX=V(R.K-1) :
        V(R.K-1)=V(R.K) :
        V(R.K)=R.AUX :
        R.BUN=0
1931     NEXT
1932   WEND
1935   RETURN
1936   WEND
MERGE "C7"
Ok
MERGE "S4"
```

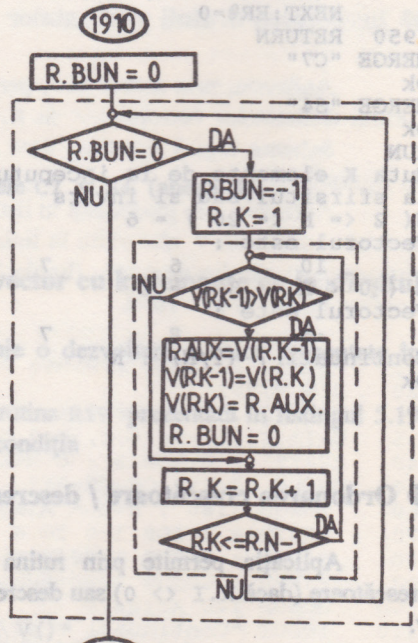


Fig. 5.17

```
Ok
RUN
Ordonarea crescatoare/descrescatoare a lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10          6          7          3         -6          8
Crescator/descrescator (*/0) : 0
Vectorul este :
      10          8          7          6          3         -6
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 5
Vectorul este :
      10          6          7          3         -6
Crescator/descrescator (*/0) : 1
Vectorul este :
      -6          3          6          7         10
Continuati ? (Y/N) : N
Ok
```

Listing 5.20

În realizarea aplicației s-au utilizat de asemenea rutinele `C7` și `S4` (anexa A).

● Rotirea stînga / dreapta a elementelor unui vector

Rotirea dreapta cu 3 poziții a elementelor unui vector `V` este prezentată schematic în figura 5.18. Rotirea s-a efectuat pas cu pas astfel încît elementul ajuns în extremitatea dreaptă este reținut pînă la deplasarea către dreapta cu o poziție a întregului vector, după care acesta trece pe prima poziție din stînga, și așa mai departe.

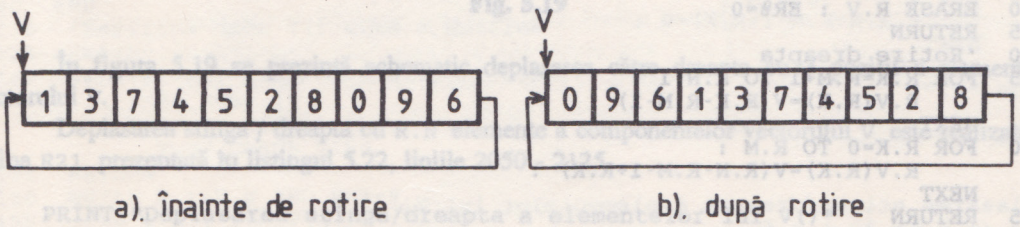


Fig. 5.18

Rotirea stînga / dreapta cu `R.I` elemente a vectorului `V` este efectuată de rutina `R20`. Dacă `R.I < 0` se efectuează rotirea stînga a vectorului `V` cu `R.I` poziții, altfel rotirea se efectuează către dreapta. Rutina este prezentată în listingul 5.21, liniile 1960 - 2045.

```
10    PRINT "Rotirea stinga/dreapta a elementelor lui V()"
20    DIM V(20)
30    DATA 10,6,7,3,-6,8,9,-11,6,24
35    DATA 4,11,-3,-8,0,-9,2,1,-1,21
```

```
40    C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50    WHILE C$="Y" OR C$="Y"
60     GOSUB 3180 : N=C.A'Alege N
70     RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80     S.N=N : GOSUB 5100'Scrie vectorul
90     INPUT "I = ",R.I
100    R.N=N : GOSUB 1960'Rotire
110    GOSUB 150'Afisare
120    INPUT "Continuati ? (Y/N) : ",C$
130   WEND
140   END
150    'Afisare
160    IF ER%<>0
            THEN PRINT "ER =";ER%
            ELSE S.N=N : GOSUB 5100
170    RETURN
1960   'Rotirea stinga/dreapta a lui V()
1965   IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
           THEN ER%=-5
           ELSE IF R.I=0
                 THEN ER%=0
                 ELSE GOSUB 1975
1970   RETURN
1975   DIM R.V(20)
1980   R.J=ABS(R.I)
1985   IF R.J<R.N
           THEN R.M=R.J-1
           ELSE R.M=R.J MOD R.N
1990   IF R.I > 0
           THEN GOSUB 2010
           ELSE GOSUB 2030
1995   FOR R.K=0 TO R.N-1 :
           V(R.K)=R.V(R.K) :
       NEXT
2000   ERASE R.V : ER%=0
2005   RETURN
2010   'Rotire dreapta
2015   FOR R.K=R.M+1 TO R.N-1 :
           R.V(R.K)=V(R.K-R.M-1) :
       NEXT
2020   FOR R.K=0 TO R.M :
           R.V(R.K)=V(R.N-R.M-1+R.K) :
       NEXT
2025   RETURN
2030   'Rotire stinga
2035   FOR R.K=R.M+1 TO R.N-1 :
           R.V(R.K-R.M-1)=V(R.K) :
       NEXT
2040   FOR R.K=0 TO R.M :
           R.V(R.N-R.M-1+R.K)=V(R.K) :
       NEXT
2045   RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Rotirea stinga/dreapta a elementelor lui V()
N( 2 <= N <= 20 ) = 6
```

```
Vectorul este :
      10        6         7         3        -6         8
I = 2
Vectorul este :
      -6        8        10         6         7         3
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 6
Vectorul este :
      10        6         7         3        -6         8
I = -2
Vectorul este :
       7        3        -6         8        10         6
Continuati ? (Y/N) : N
Ok
```

Listing 5.21

În realizarea aplicației s-au mai utilizat și rutinele `C7` și `S4` (anexa A).

● Deplasarea stînga / dreapta a elementelor unui vector

Spre deosebire de rotire, deplasarea implică pierderea elementelor care datorită mutării în direcția dorită ies în afară; pozițiile rămase libere ca urmare a deplasării capătă automat valoarea 0.

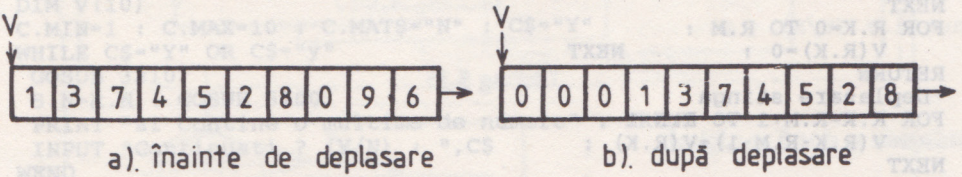


Fig. 5.19

În figura 5.19 se prezintă schematic deplasarea către dreapta cu 3 poziții a elementelor vectorului `V`.

Deplasarea stînga / dreapta cu `R.N` elemente a componentelor vectorului `V` este realizată de rutina `R21` prezentată în listingul 5.22, liniile 2050 - 2125.

```
10   PRINT "Deplasarea stinga/dreapta a elementelor lui V()"
20   DIM V(20)
30   DATA 10,6,7,3,-6,8,9,-11,6,24
35   DATA 4,11,-3,-8,0,-9,2,1,-1,21
40   C$="Y" : C.MIN=2 : C.MAX=20 : C.MAT$="N"
50   WHILE C$="Y" OR C$="Y"
60    GOSUB 3180 : N=C.A'Alege N
70    RESTORE : FOR I=0 TO N-1 : READ V(I) : NEXT
80    S.N=N : GOSUB 5100'Scrie vectorul
90    INPUT "I = ",R.I
100   R.N=N : GOSUB 2050'Deplasare
110   GOSUB 150'Afisare
120   INPUT "Continuati ? (Y/N) : ",C$
130  WEND
140  END
```

```
150   'Afisare
160   IF ER%<>0
        THEN PRINT "ER =";ER%
        ELSE S.N=N : GOSUB 5100
170   RETURN
2050  'Deplasare stinga/dreapta a lui V()
2055  IF R.N<>INT(R.N) OR R.N<2 OR R.N>20
        THEN ER%=-5
        ELSE IF R.I<>INT(R.I)
               THEN ER%=-2
               ELSE IF R.I=0
                      THEN ER%=0
                      ELSE GOSUB 2065
2060  RETURN
2065  R.J=ABS(R.I)
2070  IF R.J<R.N
        THEN R.M=R.J-1
        ELSE R.M=R.J MOD R.N
2075  IF R.I > 0
        THEN GOSUB 2090
        ELSE GOSUB 2110
2080  ER%=0
2085  RETURN
2090  'Deplasare dreapta
2095  FOR R.K=R.N-1 TO R.M+1 STEP -1 :
        V(R.K)=V(R.K-R.M-1) :
      NEXT
2100  FOR R.K=0 TO R.M :
        V(R.K)=0 :          NEXT
2105  RETURN
2110  'Deplasare stinga
2115  FOR R.K=R.M+1 TO R.N-1 :
        V(R.K-R.M-1)=V(R.K) :
      NEXT
2120  FOR R.K=0 TO R.M :
        V(R.N-R.M-1+R.K)=0 :
      NEXT
2125  RETURN
MERGE "C7"
Ok
MERGE "S4"
Ok
RUN
Deplasarea stinga/dreapta a elementelor lui V()
N( 2 <= N <= 20 ) = 6
Vectorul este :
       10          6          7          3         -6          8
I = 2
Vectorul este :
        0          0         10          6          7          3
Continuati ? (Y/N) : Y
N( 2 <= N <= 20 ) = 6
Vectorul este :
       10          6          7          3         -6          8
I = -2
Vectorul este :
        7          3         -6          8          0          0
Continuati ? (Y/N) : N
Ok
```

Listing 5.22

5.3 Mulţimi

● Validarea unei mulţimi de numere

Într-o mulţime orice element este unic. În consecinţă validarea unei mulţimi de numere presupune verificarea unicităţii fiecărui element al său. Fie un vector V conţinînd R.N valori numerice. *Validarea* se efectuează astfel:

- *se ordonează crescător elementele acestuia;*
- *se verifică existenţa a cel puţin două elemente cu valori egale şi:*
  - *dacă există* atunci vectorul V *nu conţine o mulţime* de numere;
  - *dacă nu* atunci vectorul V *conţine o mulţime* de numere.

În cadrul aplicaţiei prezentate în listingul 5.23, ordonarea elementelor vectorului V este efectuată utilizînd rutina R18 (anexa A), iar verificarea unicităţii valorilor elementelor acestuia cu rutina R22 (listingul 5.23, liniile 2130 - 2170). În vederea simplificării construirii de aplicaţii care operează cu mulţimi de numere, rutinele R18 şi R22 au fost cuplate împreună cu C7 în rutina C8 (listingul 5.23, liniile 3210 - 3260) care permite citirea, ordonarea crescătoare şi validarea unei mulţimi de numere întregi.

```
10    PRINT "Citeste si verifica o multime de numere"
20    DIM V(10)
30    C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40    WHILE C$="Y" OR C$="y"
50     GOSUB 3210
60     S.N=R.N : GOSUB 5100
70     PRINT "si contine o multime de numere" : PRINT
80     INPUT "Continuati ? (Y/N) : ",C$
90    WEND
100   END
2130  'Verifica daca V() este o multime
2135  ER%=0 : R.I=1
2140  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10
        THEN ER%=-11
        ELSE IF R.N => 2
          THEN GOSUB 2150
2145  RETURN
2150  WHILE R.I<R.N AND ER%=0
2155    IF V(R.I-1)=>V(R.I)
          THEN ER%=-10
2160    R.I=R.I+1
2165  WEND
2170  RETURN
3210  'Citeste si valideaza o multime V()
3215  ER%=-10
3220  WHILE ER%<>0
3225   GOSUB 3180 : R.N=C.A'Citeste N
3230   PRINT "Introduceti multimea de";R.N;"numere :"
3235   FOR R.I=1 TO R.N :
       PRINT "   N";:PRINT USING "$#";R.I;:
       INPUT;"= ",V(R.I-1):
        NEXT : PRINT
3240   R.I=1 : GOSUB 1860'Ordonare
```

```
3245   GOSUB 2130'Verificare
3250   IF ER%<>0
            THEN IF ER%=-10
                 THEN PRINT "ER : nu este o multime"
                 ELSE PRINT "ER :";ER%
3255   WEND
3260   RETURN
MERGE "C7"
Ok
MERGE "R18"
Ok
MERGE "S4"
Ok
RUN
Citeste si valideaza o multime de numere
N( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :
   N$1= 1   N$2= 0   N$3= 8   N$4= 5   N$5= 7   N$6= 0
ER : nu este o multime
N( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :
   N$1= 1   N$2= 0   N$3= 8   N$4= 5   N$5= 7   N$6= 4
Vectorul este :
     0         1         4         5         7         8
si contine o multime

Continuati ? (Y/N) : N
Ok
```

Listing 5.23

● Reuniunea a două mulţimi

Aplicaţia permiteă, cu ajutorul rutinei `R23`, obţinerea reuniunii mulţimilor de numere `A` şi `B`.

Schema logică a rutinei `R23` este prezentată în figura 5.20. Variabilele de intrare în rutină sînt `R.N` (numărul elementelor mulţimii `A`) şi `R.M` (numărul elementelor mulţimii `B`). Vectorii `A` şi `B` respectiv `C` (conţine rezultatul reuniunii) sînt ordonaţi crescător. Numărul elementelor mulţimii `C` este atribuit variabilei `R.K`.

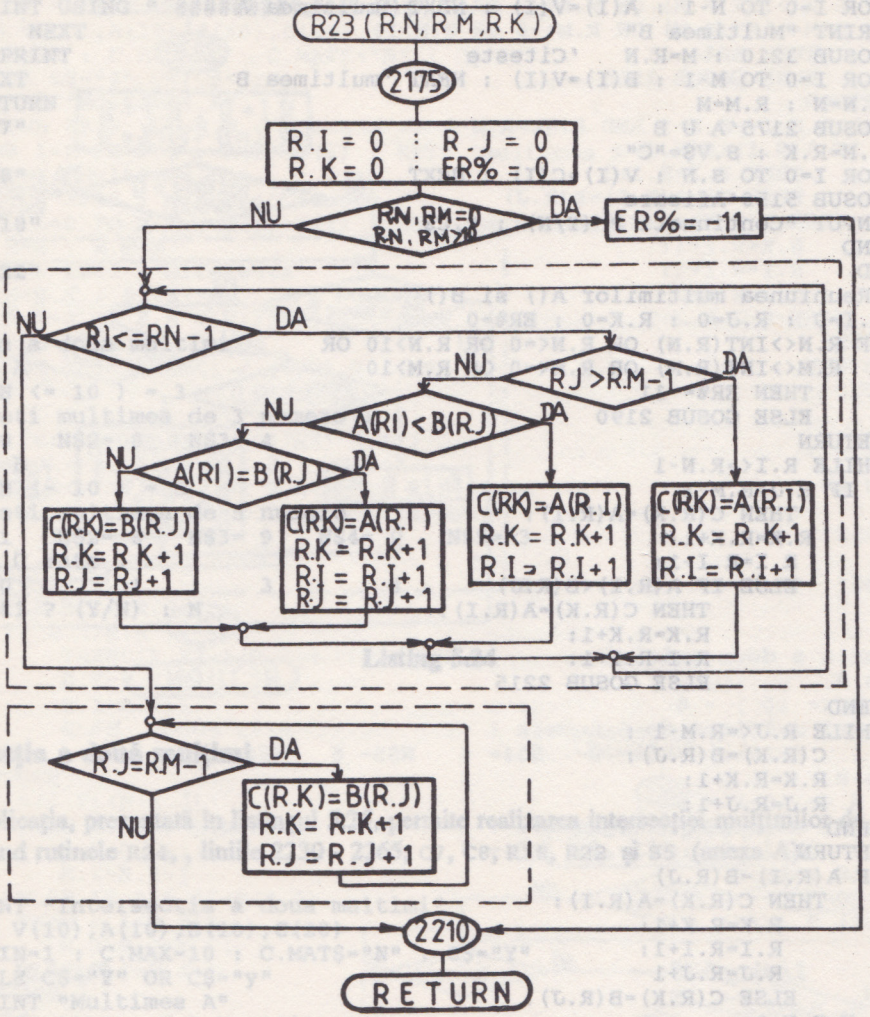


Fig. 5.20

Aplicaţia, prezentată în listingul 5.24, utilizează pe lîngă rutina `R23` **(liniile 2175 - 2220) şi** `S5` **(liniile 5150 - 5195), rutinele** `C7`, `C8`, `R18` **şi** `R22` **(anexa A).**

```
10    PRINT "Reuniunea a doua multimi"
20    DIM V(10),A(10),B(10),C(20)
30    C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$-"Y"
40    WHILE C$="Y" OR C$="y"
50     PRINT "Multimea A"
60     GOSUB 3210 : N=R.N   'Citeste
```

```
70    FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80    PRINT "Multimea B"
90    GOSUB 3210 : M=R.N   'Citeste
100   FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110   R.N=N : R.M=M
120   GOSUB 2175'A U B
130   S.N=R.K : S.V$="C"
140   FOR I=0 TO S.N : V(I)=C(I) : NEXT
150   GOSUB 5150'Afisare
160   INPUT "Continuati ? (Y/N) : ",C$
170  WEND
180  END
2175  'Reuniunea multimilor A() si B()
2180  R.I=0 : R.J=0 : R.K=0 : ER%=0
2185  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
         R.M<>INT(R.M) OR R.M<=0 OR R.M>10
           THEN ER%=-11
           ELSE GOSUB 2190
2186  RETURN
2190  WHILE R.I<=R.N-1
2195    IF R.J>R.M-1
           THEN C(R.K)=A(R.I):
           R.K=R.K+1:
           R.I=R.I+1:
           ELSE IF A(R.I)<B(R.J)
                  THEN C(R.K)=A(R.I):
                  R.K=R.K+1:
                  R.I=R.I+1:
                  ELSE GOSUB 2215
2200  WEND
2205  WHILE R.J<=R.M-1 :
        C(R.K)=B(R.J):
        R.K=R.K+1:
        R.J=R.J+1:
      WEND
2210  RETURN
2215  IF A(R.I)=B(R.J)
        THEN C(R.K)=A(R.I):
          R.K=R.K+1:
          R.I=R.I+1:
          R.J=R.J+1
        ELSE C(R.K)=B(R.J):
      R.K=R.K+1:
      R.J=R.J+1
2220  RETURN
5150  'Scrie un vector V() de S.N elemente
5155  IF INT(S.N)<>S.N OR S.N>20 OR S.N<=0
        THEN PRINT "ER : indice ";S.N;" eronat " :
        GOTO 5195
5160  IF ER%<>0
        THEN PRINT "ER% =";ER% : GOTO 5195
5165  PRINT "Multimea ";S.V$;" este :"
5170  FOR S.I=0 TO S.N-1 STEP 7
5175    IF S.I+6 <= S.N-1
          THEN S.F=S.I+6
          ELSE S.F=S.N-1
5180    FOR S.K=S.I TO S.F :
```

```
     PRINT USING " ########";V(S.K); :
        NEXT
5185    PRINT
5190  NEXT
5195  RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
RUN
Reuniunea a doua multimi
Multimea A
N( 1 <= N <= 10 ) = 3
Introduceti multimea de 3 numere :
   N$1= 0   N$2= 8   N$3= 4
Multimea B
N( 1 <= N <= 10 ) = 5
Introduceti multimea de 5 numere :
   N$1= 1   N$2= 8   N$3= 9   N$4= 0   N$5= 3
Multimea C este :
      0        1        3        4        8        9
Continuati ? (Y/N) : N
Ok
```

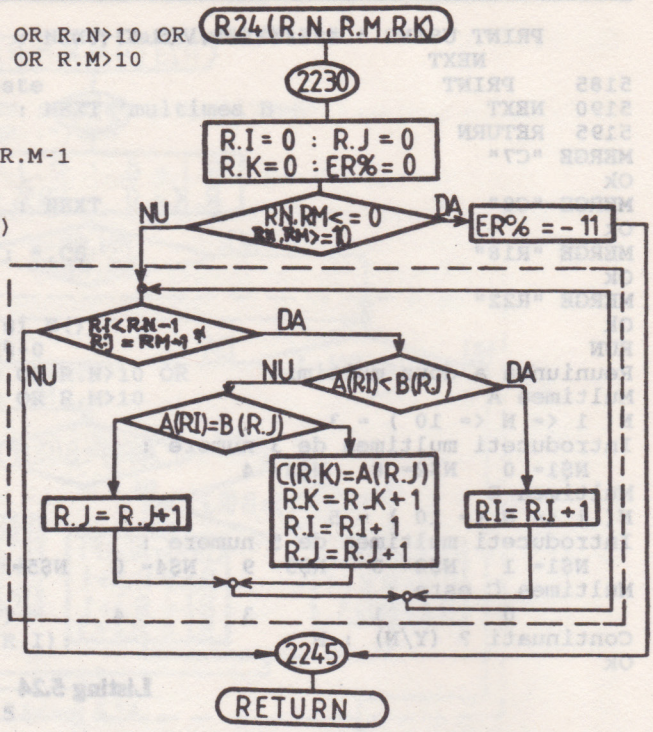
Listing 5.24

● Intersecţia a două mulţimi

Aplicaţia, prezentată în listingul 5.25, permite realizarea intersecţiei mulţimilor de numere A şi B utilizînd rutinele R24, , liniile 2230 - 2265, C7, C8, R18, R22 şi S5 (anexa A).

```
10   PRINT "Intersectia a doua multimi"
20   DIM V(10),A(10),B(10),C(20)
30   C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40   WHILE C$="Y" OR C$="y"
50    PRINT "Multimea A"
60    GOSUB 3210 : N=R.N  'Citeste
70    FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80    PRINT "Multimea B"
90    GOSUB 3210 : M=R.N  'Citeste
100   FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110   R.N=N : R.M=M
120   GOSUB 2230  'Intersectie
130   S.N=R.K : S.V$="C"
140   FOR I=0 TO S.N : V(I)=C(I) : NEXT
150   GOSUB 5150  'Afisare
160   INPUT "Continuati ? (Y/N) : ",C$
170  WEND
180  END
2230  'Intersectia multimilor A() si B()
2235  R.I=0 : R.J=0 : R.K=0 : ER%=0
```

```
2240  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
      R.M<>INT(R.M) OR R.M<=0 OR R.M>10
        THEN ER%=-11
        ELSE GOSUB 2250
2245  RETURN
2250  WHILE R.I<=R.N-1 AND R.J<=R.M-1
2255    IF A(R.I)<B(R.J)
          THEN R.I=R.I+1
          ELSE IF A(R.I)=B(R.J)
        THEN C(R.K)=A(R.I):
             R.K=R.K+1:
             R.I=R.I+1:
             R.J=R.J+1
        ELSE R.J=R.J+1
2260  WEND
2265  RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
MERGE "S5"
Ok
RUN
Intersectia a doua multimi
Multimea A
N( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :
   N$1= 1   N$2= 4   N$3= 0   N$4= 8   N$5= 6   N$6= 2
Multimea B
N( 1 <= N <= 10 ) = 4
Introduceti multimea de 4 numere :
   N$1= 3   N$2= 8   N$3= 0   N$4= 7
Multimea C este :
        0          8
Continuati ? (Y/N) : N
Ok
```



Listing 5.25 **Fig. 5.21**

Rutina R24 realizează intersecţia mulţimilor ordonate A şi B avînd R.N respectiv R.M elemente. Rezultatul este depus în vectorul C iar numărul elementelor acesteia este atribuit variabilei R.K. Schema logică a rutinei R24 este prezentată în figura 5.21.

● Diferenţa a două mulţimi

Fie A şi B două mulţimi de R.N respectiv R.M elemente. Mulţimea C = A \ B va conţine toate elementele care aparţin de A şi nu aparţin de B.

Diferenţa mulţimilor A şi B este realizată de rutina R25 a cărei schemă logică este prezentată în figura 5.22. Textul sursă al rutinei este prezentat în listingul 5.26, liniile 2280 - 2325.

```
10   PRINT "Diferenta a doua multimi"
20   DIM V(10),A(10),B(10),C(20)
30   C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40   WHILE C$="Y" OR C$="y"
50    PRINT "Multimea A"
60    GOSUB 3210 : N=R.N  'Citeste
70    FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80    PRINT "Multimea B"
90    GOSUB 3210 : M=R.N  'Citeste
100   FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110   R.N=N : R.M=M
120   GOSUB 2280  'Diferenta
130   S.N=R.K : S.V$="C"
140   FOR I=0 TO S.N : V(I)=C(I) : NEXT
150   GOSUB 5150  'Afisare
160   INPUT "Continuati ? (Y/N) : ",C$
170  WEND
180  END
2280  'Diferenta multimilor A() \ B()
2285  R.I=0 : R.J=0 : R.K=0 : ER%=0
2290  IF R.N<>INT(R.N) OR R.N<=0 OR R.N>10 OR
        R.M<>INT(R.M) OR R.M<=0 OR R.M>10
          THEN ER%=-11
          ELSE GOSUB 2300
2295  RETURN
2300  WHILE R.I<=R.N-1
2305    IF R.J>R.M
          THENC(R.K)=A(R.I):
          R.K=R.K+1:
          R.I=R.I+1
          ELSE  GOSUB 2320
2310  WEND
2315  RETURN
2320  IF A(R.I)<B(R.J)
        THEN C(R.K)=A(R.I):
          R.K=R.K+1:
          R.I=R.I+1
        ELSE IF A(R.I)=B(R.J)
        THEN R.I=R.I+1:
          R.J=R.J+1
        ELSE R.J=R.J+1
2325  RETURN
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
MERGE "S5"
Ok
RUN
Diferenta a doua multimi
Multimea A
N( 1 <= N <= 10 ) = 6
Introduceti multimea de 6 numere :
```

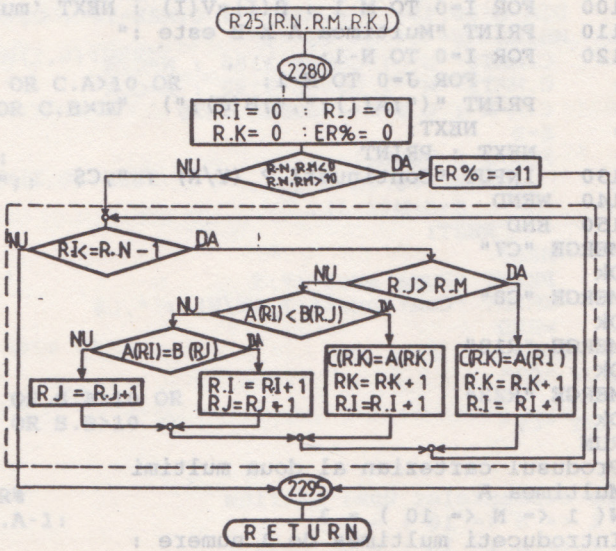


Fig. 5.22

```
   N$1= 1   N$2= 2   N$3= 5   N$4= 7   N$5= 9   N$6= 8
Multimea B
N( 1 <= N <= 10 ) = 3
Introduceti multimea de 3 numere :
   N$1= 5   N$2= 0   N$3= 8
Multimea C este :
        1         2         7         9
Continuati ? (Y/N) : N
Ok
```

Listing 5.26

● Produsul cartezian al două mulţimi de numere

Aplicaţia permite citirea, ordonarea şi validarea mulţimilor A şi B de N respectiv M elemente şi determină produsul cartezian A X B fără a reţine însă mulţimea perechilor formate.

Aplicaţia este prezentată în listingul 5.27.

```
10    PRINT "Produsul cartezian al doua multimi"
20    DIM V(10),A(10),B(10)
30    C.MIN=1 : C.MAX=10 : C.MAT$="N" : C$="Y"
40    WHILE C$="Y" OR C$="y"
50     PRINT "Multimea A"
60     GOSUB 3210 : N=R.N  'Citeste
70     FOR I=0 TO N-1 : A(I)=V(I) : NEXT 'multimea A
80     PRINT "Multimea B"
90     GOSUB 3210 : M=R.N  'Citeste
100    FOR I=0 TO M-1 : B(I)=V(I) : NEXT 'multimea B
110    PRINT "Multimea A X B este :"
120    FOR I=0 TO N-1:
           FOR J=0 TO M-1:
       PRINT "(";A(I);",";B(J);")  ";:
           NEXT:
       NEXT : PRINT
130    INPUT "Continuati ? (Y/N) : ",C$
140   WEND
150   END
MERGE "C7"
Ok
MERGE "C8"
Ok
MERGE "R18"
Ok
MERGE "R22"
Ok
RUN
Produsul cartezian al doua multimi
Multimea A
N( 1 <= N <= 10 ) = 3
Introduceti multimea de 3 numere :
   N$1= 1   N$2= 4   N$3= 6
Multimea B
N( 1 <= N <= 10 ) = 2
Introduceti multimea de 2 numere :
```

```
   N$1= 1   N$2= 5
Multimea A X B este :
( 1 , 1 )  ( 1 , 5 )  ( 4 , 1 )  ( 4 , 5 )  ( 6 , 1 )  ( 6 , 5 )
Continuati ? (Y/N) : N
Ok
```

Listing 5.27

5.4 Matrici, determinanţi

● Citirea şi listarea unei matrice

Aplicaţia oferă o variantă de rutine pentru citirea şi validarea elementelor unei matrici de numere întregi (rutina C9) precum şi pentru listarea acesteia utilizînd rutina S6. Prezentată în listingul 5.28 aplicaţia conţine alături de programul principal (liniile 10 - 110), rutinele C9 (liniile 3270 - 3280) şi S6 (liniile 5200 - 5210).

```
10   PRINT "Citeste si listeaza o matrice"
20   DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30   WHILE C$="Y" OR C$="y"
40    C.MAT$="N" : GOSUB 3180 : N=C.A
50    C.MAT$="M" : GOSUB 3180 : M=C.A
60    C.A=N : C.B=M : GOSUB 3270
70    PRINT "Matricea M este:"
80    S.A=N : S.B=M : GOSUB 5200
90    INPUT "Continuati ? (Y/N) : ",C$
100  WEND
110  END
3270  'Citeste matricea M()
3275  IF C.A<>INT(C.A) OR C.A<1 OR C.A>10 OR
        C.B<>INT(C.B) OR C.B<1 OR C.B>10
        THEN ER%=-12
        ELSE FOR C.I=0 TO C.A-1:
         PRINT "Linia";C.I+1;":";:
         FOR C.J=0 TO C.B-1:
      INPUT;" ",M(C.I,C.J):
      PRINT ",";:
         NEXT:
         PRINT:
      NEXT
3280  RETURN
5200  'Afiseaza matricea M()
5205  IF S.A<>INT(S.A) OR S.A<1 OR S.A>10 OR
        S.B<>INT(S.B) OR S.B<1 OR S.B>10
        THEN PRINT "ER = -12"
        ELSE IF ER%<>0
           THEN PRINT "ER =",ER%
           ELSE FOR S.I=0 TO S.A-1:
          FOR S.J=0 TO S.B-1:
           PRINT M(S.I,S.J);:
          NEXT:PRINT:
      NEXT
5210  RETURN
```

```
MERGE "C7"
Ok
RUN
Citeste si listeaza o matrice
N( 1 <= N <= 10 ) = 3
M( 1 <= N <= 10 ) = 5
Linia 1 : 1, 4, 6, 7, 8,
Linia 2 : 9, 0, 0, 8, 9,
Linia 3 : 1, 3, 4, 6, 7,
Matricea M este :
 1  4  6  7  8
 9  0  0  8  9
 1  3  4  6  7
Continuati ? (Y/N) : N
Ok
```

Listing 5.28

Pentru citirea şi validarea variabilelor N (numărul de linii) şi M (numărul de coloane) care definesc dimensiunile matricei M este utilizată rutina C7.

● Suma elementelor unei matrice

În realizarea acestei aplicaţii, prezentată în listingul 5.29, s-au utilizat rutinele C7 (pentru citirea şi validarea unui număr întreg cuprins între C.MIN şi C.MAX) şi C9 (citirea şi validarea elementelor unei matrici de C.A X C.B componente).

```
10    PRINT "Suma elementelor unei matrice"
20    DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30    WHILE C$="Y" OR C$="y"
40     C.MAT$="N" : GOSUB 3180 : N=C.A
50     C.MAT$="M" : GOSUB 3180 : M=C.A
60     C.A=N : C.B=M : GOSUB 3270
70     S=0
80     FOR I=0 TO N-1:
           FOR J=0 TO M-1:
             S=S+M(I,J):
           NEXT:
       NEXT
90     PRINT "Suma este :";S
100    INPUT "Continuati ? (Y/N) : ",C$
110    WEND
120   END
MERGE "C7"
Ok
MERGE "C9"
Ok
RUN
Suma elementelor unei matrice
N( 1 <= N <= 10 ) = 5
M( 1 <= N <= 10 ) = 6
Linia 1 : 1, 4, 8, 9, 0, 2,
Linia 2 : 3, 2, 1, 7, 0, 6,
Linia 3 : 5, 8, 9, 2, 4, 3,
```

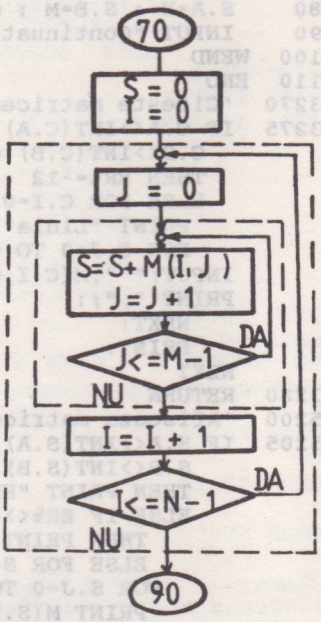


Fig. 5.23

```
Linia 4 : 7, 8, 9, 1, 3, 4,
Linia 5 : 4, 6, 7, 7, 2, 9,
Suma este : 141
Continuati ? (Y/N) : N
Ok
```

Listing 5.29

Suma propriu-zisă a elementelor matricei `M` este realizată în secvenţa de program cuprinsă între liniile 70 la 90. Schema logică a secvenţei este prezentată în figura 5.23. Variabilele `N` şi `M` reprezintă dimensiunile matricei `M` iar variabila `S` conţine suma acestora.

● Suma a două matrici

Aplicaţia, prezentată în listingul 5.30, permite citirea şi validarea elementelor matricilor `A` şi `B` de `N X M` componente (liniile 50 - 120), determinarea sumei acestora (linia 130) precum şi afişarea matricei sumă `M` (liniile 140 - 150).

```
10    PRINT "Suma a doua matrici (n X m)"
20    DIM M(10,10),A(10,10),B(10,10)
30    C.MIN=1 : C.MAX=10 : C$="Y"
40    WHILE C$="Y" OR C$="y"
50     C.MAT$="N" : GOSUB 3180 : N=C.A
60     C.MAT$="M" : GOSUB 3180 : M=C.A
70     PRINT "Matricea A :"
80     C.A=N : C.B=M : GOSUB 3270
90     FOR I=0 TO N-1 :
           FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT:
       NEXT
100    PRINT "Matricea B :"
110    GOSUB 3270
120    FOR I=0 TO N-1 :
           FOR J=0 TO M-1:B(I,J)=M(I,J):NEXT:
       NEXT
130    FOR I=0 TO N-1 :
           FOR J=0 TO M-1:
             M(I,J)=A(I,J)+B(I,J):
           NEXT:
       NEXT
140    PRINT "Matricea M = A + B este :"
150    S.A=N : S.B=M : GOSUB 5200
160    INPUT "Continuati ? (Y/N) : ",C$
170   WEND
180   END
MERGE "C7"
Ok
MERGE "C9"
Ok
MERGE "S6"
Ok
RUN
Suma a doua matrici (n X m)
```

```
N( 1 <= N <= 10 ) = 3
M( 1 <= N <= 10 ) = 4
Matricea A :
Linia 1 : 8, 9, 6, 4,
Linia 2 : 2, 6, 8, 5,
Linia 3 : 5, 7, 2, 1,
Matricea B :
Linia 1 : 9, 7, 7, 8,
Linia 2 : 8, 4, 3, 9,
Linia 3 : 7, 9, 8, 9,
Matricea M = A + B este :
 17  16  13  12
 10  10  11  14
 12  16  10  10
Continuati ? (Y/N) : N
Ok
```

Listing 5.30

Observaţie. *Deoarece citirea şi validarea elementelor matricilor* `A` *şi* `B` *s-a efectuat cu rutina* `C9` *(anexa A) care depune valorile elementelor citite în matricea* `M` *de* `C.A` *X* `C.B` *elemente, după revenirea din rutină este necesară transferarea elementelor matricei* `M` *în matricea* `A` *(linia 90) respectiv* `B` *(linia 120).*

● Produsul a două matrici

Aplicaţia (listingul 5.31) permite determinarea produsului matricilor `A` şi `B` de (`N` X `M`) respectiv (`M` X `P`) elemente. Rezultatul este stocat în matricea `M` de (`N` X `P`) elemente.

Citirea şi validarea elementelor matricilor `A` şi `B` se efectuează utilizînd rutinele `C7` şi `C9` (anexa A). Produsul efectiv al matricilor `A` şi `B` se realizează în secvenţa din linia 150 (listingul 5.31).

```
10    PRINT "Produsul a doua matrici"
20    PRINT "A(n X m) si B(m X p)"
30    DIM M(10,10),A(10,10),B(10,10)
40    C.MIN=1 : C.MAX=10 : C$="Y"
50    WHILE C$="Y" OR C$="y"
60     C.MAT$="N" : GOSUB 3180 : N=C.A
70     C.MAT$="M" : GOSUB 3180 : M=C.A
80     C.MAT$="P" : GOSUB 3180 : P=C.A
90     PRINT "Matricea A(";N;"X";M;") :"
100    C.A=N : C.B=M : GOSUB 3270
110    FOR I=0 TO N-1 :
           FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT:
       NEXT
120    PRINT "Matricea B(";M;"X";P;") :"
130    C.A=M : C.B=P : GOSUB 3270
140    FOR I=0 TO M-1 :
           FOR J=0 TO P-1:B(I,J)=M(I,J):NEXT:
       NEXT
150    FOR I=0 TO N-1:
           FOR J=0 TO P-1:
             M(I,J)=0:
```

```
                FOR K=0 TO M-1:
                M(I,J)=M(I,J)+A(I,K)*B(K,J):
                NEXT:
          NEXT:
       NEXT
160    PRINT "Matricea M = A X B este :"
170    S.A=N : S.B=P : GOSUB 5200
180    INPUT "Continuati ? (Y/N) : ",C$
190   WEND
200   END
MERGE "C7"
Ok
MERGE "C9"
Ok
MERGE "S6"
Ok
RUN
Produsul a doua matrici
A(n X m) si B(m X p)
N( 1 <= N <= 10 ) = 4
M( 1 <= N <= 10 ) = 3
P( 1 <= P <= 10 ) = 5
Matricea A( 4 X 3 ) :
Linia 1 : 1, 4, 3,
Linia 2 : 2, 1, 5,
Linia 3 : 3, 2, 3,
Linia 4 : 2, 1, 4,
Matricea B( 3 X 5 ) :
Linia 1 : 2, 5, 7, 6, 0,
Linia 2 : 3, 1, 4, 5, 2,
Linia 3 : 2, 8, 1, 3, 3,
Matricea M = A X B este :
 20   33   26   35   17
 17   51   23   32   17
 18   41   32   37   13
 15   43   22   29   14
Continuati ? (Y/N) : N
Ok
```

Listing 5.31

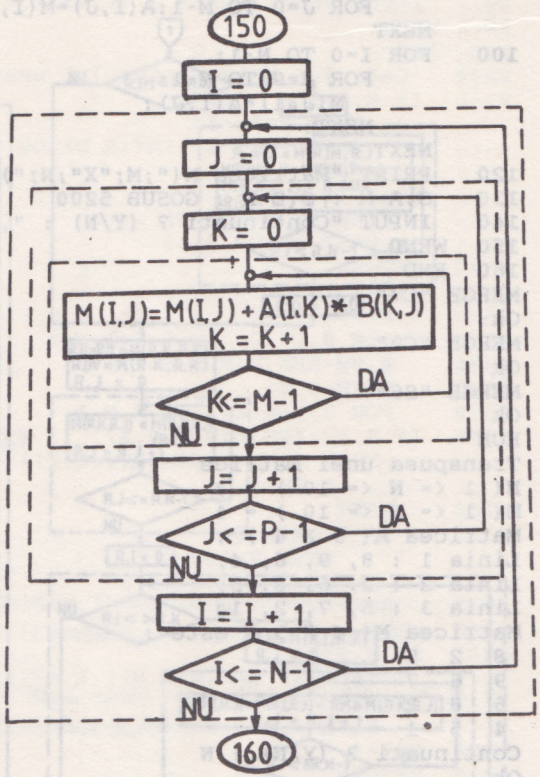


Fig. 5.24

Schema logică a acestei secvenţe este prezentată în figura 5.24.

● Transpusa unei matrice

Aplicaţia este prezentată în listingul 5.32. Pentru determinarea transpusei matricei `M` de (`N X M`) componente s-a utilizat matricea auxiliară `A`. În realizarea aplicaţiei s-au utilizat de asemenea rutinele `C7`, `C9` şi `S6` (anexa A).

```
10    PRINT "Transpusa unei matrice"
20    DIM M(10,10),A(10,10)
30    C.MIN=1 : C.MAX=10 : C$="Y"
40    WHILE C$="Y" OR C$="y"
50     C.MAT$="N" : GOSUB 3180 : N=C.A
60     C.MAT$="M" : GOSUB 3180 : M=C.A
70     PRINT "Matricea A(";N;"X";M;") :"
```

```
80    C.A=N : C.B=M : GOSUB 3270
90    FOR I=0 TO N-1 :
          FOR J=0 TO M-1:A(I,J)=M(I,J):NEXT:
      NEXT
100   FOR I=0 TO N-1:
          FOR J=0 TO M-1:
            M(J,I)=A(I,J):
          NEXT:
      NEXT
120   PRINT "Matricea M(";M;"X";N;") este :"
130   S.A=M : S.B=N : GOSUB 5200
140   INPUT "Continuati ? (Y/N) : ",C$
150  WEND
160  END
MERGE "C7"
Ok
MERGE "C9"
Ok
MERGE "S6"
Ok
RUN
Transpusa unei matrice
N( 1 <= N <= 10 ) = 3
M( 1 <= N <= 10 ) = 4
Matricea A( 3 X 4 ) :
Linia 1 : 8, 9, 6, 4,
Linia 2 : 2, 6, 8, 5,
Linia 3 : 5, 7, 2, 1,
Matricea M( 4 X 3 ) este :
 8  2  5
 9  6  7
 6  8  2
 4  5  1
Continuati ? (Y/N) : N
Ok
```

Listing 5.32

● Calculul determinantului unei matrice

Aplicaţia permite determinarea cu o aproximaţie de 1/10000 a determinantului unei matrice pătrate de ordin N (0 < N <= 10). Calculul valorii determinantului matricei M este realizat de rutina R26 a cărei schemă logică este prezentată în figura 5.25.

Variabilele de intrare în subrutină sînt R.EPS (conţinînd valoarea aleasă pentru eroarea admisă) şi R.N (ordinul matricei). Matricea M de rang R.N se consideră variabila globală. Valoarea determinantului este depusă în variabila de ieşire R.D. În cazul cînd valoarea determinantului matricei M este mai mică decît R.EPS aceasta se consideră 0.

Aplicaţia este prezentată în listingul 5.33. Liniile 2330 - 2450 aparţin matricei R26.

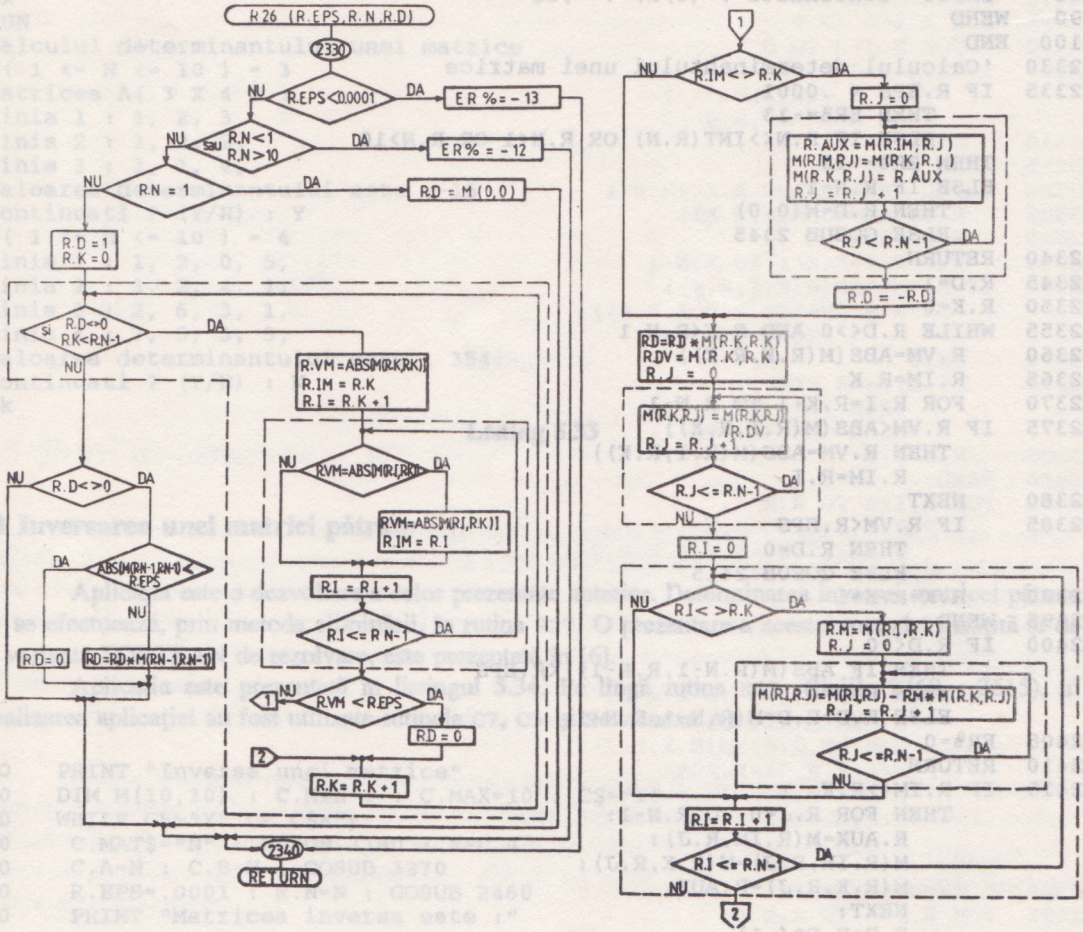


Fig. 5.25

```
10   PRINT "Calculul determinantului unei matrice"
20   DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30   WHILE C$="Y" OR C$="y"
40    C.MAT$="N" : GOSUB 3180 : N=C.A
50    C.A=N : C.B=N : GOSUB 3270
60    R.EPS=.0001 : R.N=N : GOSUB 2330
70    IF ER%=0
        THEN PRINT "Valoarea determinantului este :";
```

```
                R.D
      ELSE PRINT "ER =",ER%
80    INPUT "Continuati ? (Y/N) : ",C$
90   WEND
100  END
2330  'Calculul determinantului unei matrice
2335  IF R.EPS < .0001
        THEN ER%=-13
        ELSE IF R.N<>INT(R.N) OR R.N<1 OR R.N>10
      THEN ER%=-12
      ELSE IF R.N=1
        THEN R.D=M(0,0)
        ELSE GOSUB 2345
2340  RETURN
2345  R.D=1
2350  R.K=0
2355  WHILE R.D<>0 AND R.K<R.N-1
2360    R.VM=ABS(M(R.K,R.K))
2365    R.IM=R.K
2370    FOR R.I=R.K+1 TO R.N-1
2375  IF R.VM<ABS(M(R.I,R.K))
        THEN R.VM=ABS(M(R.I,R.K)):
          R.IM=R.I
2380    NEXT
2385    IF R.VM<R.EPS
          THEN R.D=0
          ELSE GOSUB 2415
2390    R.K=R.K+1
2395  WEND
2400  IF R.D<>0
        THEN IF ABS(M(R.N-1,R.N-1))<R.EPS
        THEN R.D=0
        ELSE R.D=R.D*M(R.N-1,R.N-1)
2405  ER%=0
2410  RETURN
2415  IF R.IM<>R.K
        THEN FOR R.J=0 TO R.N-1:
          R.AUX=M(R.IM,R.J):
          M(R.IM,R.J)=M(R.K,R.J):
          M(R.K,R.J)=R.AUX:
          NEXT:
          R.D=R.D*(-1)
2420      R.D=R.D*M(R.K,R.K)
2425      R.DV=M(R.K,R.K)
2430  FOR R.J=0 TO R.N-1:
        M(R.K,R.J)=M(R.K,R.J)/R.DV:
      NEXT
2435  FOR R.I=0 TO R.N-1
2440    IF R.I<>R.K
          THEN R.M=M(R.I,R.K):
        FOR R.J=0 TO R.N-1:
          M(R.I,R.J)=M(R.I,R.J)-R.M*
          M(R.K,R.J):
        NEXT
2445  NEXT
2450  RETURN
MERGE "C7"
```

```
Ok
MERGE "C9"
Ok
RUN
Calculul determinantului unei matrice
N( 1 <= N <= 10 ) = 3
Matricea A( 3 X 4 ) :
Linia 1 : 1, 2, 3,
Linia 2 : 3, 2, 2,
Linia 3 : 3, 1, 4,
Valoarea determinantului este :-15
Continuati ? (Y/N) : Y
N( 1 <= N <= 10 ) = 4
Linia 1 : 1, 2, 0, 5,
Linia 2 : 3, 2, 4, 1,
Linia 3 : 2, 6, 3, 1,
Linia 4 : 7, 5, 3, 5,
Valoarea determinantului este : 354
Continuati ? (Y/N) : N
Ok
```

Listing 5.33

● Inversarea unei matrici pătrate

Aplicaţia este o dezvoltare a celor prezentate anterior. Determinarea inversei matricei pătrate `M` se efectuează, prin metoda eliminării, în rutina `R27`. O prezentare a acestei metode, însoţită şi de o variantă FORTRAN de rezolvare, este prezentată în [6].

Aplicaţia este prezentată în listingul 5.34. Pe lîngă rutina `R27` (liniile 2460 - 2615), în realizarea aplicaţiei au fost utilizate rutinele `C7`, `C5` şi `S6` (anexa A).

```
10    PRINT "Inversa unei matrice"
20    DIM M(10,10) : C.MIN=1 : C.MAX=10 : C$="Y"
30    WHILE C$="Y" OR C$="y"
40     C.MAT$="N" : GOSUB 3180 : N=C.A
50     C.A=N : C.B=N : GOSUB 3270
60     R.EPS=.0001 : R.N=N : GOSUB 2460
70     PRINT "Matricea inversa este :"
80     S.A=N : S.B=N : GOSUB 5200
90     INPUT "Continuati ? (Y/N) : ",C$
100    WEND
110    END
2460   'Inversarea unei matrice
2465   IF R.EPS < .0001
           THEN ER%=-13
           ELSE IF R.N<>INT(R.N) OR R.N<1 OR R.N>10
                   THEN ER%=-12
                   ELSE IF R.N=1
                           THEN IF ABS(M(0,0))<R.EPS
                                   THEN ER%=-14
                                   ELSE M(0,0)=1/M(0,0)
                           ELSE GOSUB 2475
2470   RETURN
2475   ER%=0
```

```
2480  R.D=1
2485  R.K=0
2490  DIM R.M(10,10)
2495  FOR R.I=0 TO 9
2500    FOR R.J=0 TO 9
2505  IF R.I=R.J
        THEN R.M(R.I,R.J)=1
        ELSE R.M(R.I,R.J)=0
2510    NEXT
2515  NEXT
2520  WHILE  ER%=0 AND R.K<=R.N-1
2525    R.VM=ABS(M(R.K,R.K))
2530    R.IM=R.K
2535    FOR R.I=R.K+1 TO R.N-1
2540  IF R.VM<ABS(M(R.I,R.K))
       THEN R.VM=ABS(M(R.I,R.K)):
            R.IM=R.I
2545    NEXT
2550    IF R.VM<R.EPS
          THEN ER%=-14
          ELSE GOSUB 2585
2555    R.K=R.K+1
2560  WEND
2565  FOR R.I=0 TO R.N-1:
        FOR R.J=0 TO R.N-1:
          M(R.I,R.J)=R.M(R.I,R.J):
        NEXT:
      NEXT
2570  ERASE R.M
2580  RETURN
2585  IF R.IM<>R.K
        THEN FOR R.J=0 TO R.N-1:
          R.AUX=M(R.IM,R.J):
          M(R.IM,R.J)=M(R.K,R.J):
          M(R.K,R.J)=R.AUX:
          R.AUX=R.M(R.IM,R.J):
          R.M(R.IM,R.J)=R.M(R.K,R.J):
          R.M(R.K,R.J)=R.AUX:
      NEXT
2590  R.DV=M(R.K,R.K)
2595  FOR R.J=0 TO R.N-1:
        M(R.K,R.J)=M(R.K,R.J)/R.DV:
        R.M(R.K,R.J)=R.M(R.K,R.J)/R.DV:
      NEXT
2600  FOR R.I=0 TO R.N-1
2605    IF R.I<>R.K
          THEN R.M=M(R.I,R.K):
          FOR R.J=0 TO R.N-1:
            M(R.I,R.J)=M(R.I,R.J)-R.M*
            M(R.K,R.J):
            R.M(R.I,R.J)=R.M(R.I,R.J)-R.M*
            R.M(R.K,R.J):
          NEXT
2610  NEXT
2615  RETURN
MERGE "C7"
Ok
```

```
MERGE "C9"
Ok
MERGE "S6"
Ok
RUN
Inversa unei matrice
N( 1 <= N <= 10 ) = 3
Linia 1 : 1, 2, 3,
Linia 2 : 2, 5, 2,
Linia 3 : 7, 3, 8,
Matricea inversa este :
-.596491   .122807   .192982
 .0350877   .22807  -.070174
 .508772  -.192982  -.017439
Continuati ? (Y/N) : N
Ok
```

Listing 5.34

5.5 Şiruri de caractere

După cum este cunoscut, unitatea de bază adresabilă în calculatoarele numerice este octetul (baitul). Cei 8 biţi ai acestuia permit alcătuirea a 256 combinaţii binare (de la 00000000 la 11111111), ceea ce oferă posibilitatea reprezentării pe un octet a tot atîtea coduri. Ca atare există posibilitatea reprezentării nu numai a codurilor ASCII (anexa B) ci şi a încă o serie de semne speciale.

Acest mod de utilizare a octetului dă posibilitatea interpretării informaţiilor memorate în calculator nu numai ca numere ci şi ca şiruri de caractere (de simboluri obţinute prin combinarea diferită a biţilor aparţinînd unui octet). Astfel un caracter este asimilat cu un octet.

Atenţie. Deoarece reprezentarea internă a informaţiilor în calculator este unică (configuraţii binare), deosebirea şi interpretarea corectă a acestora (valori numerice sau şiruri de caractere) revine utilizatorului. Pentru aceasta, aşa cum s-a arătat în capitolul 3, programatorul are la dispoziţie atît funcţii cît şi moduri de scriere a variabilelor diferite specifice celor două moduri de interpretare şi tratare a informaţiilor.

În acest subcapitol prezentăm patru aplicaţii deosebit de simple dar utile, mai ales începătorilor în programare, ca exemplificări ale unor aspecte specifice lucrului cu şiruri de caractere.

● Conversia unui număr natural (32<=N<=26) în caracter

Aplicaţia este prezentată în listingul 5.35.

```
10   PRINT "Conversii numar -> caracter"
20   C.MIN=32 : C.MAX=126 : C.MAT$="Nr." : C$="Y"
30   WHILE C$="Y" OR C$="y"
40    GOSUB 3180 : N=C.A
50    PRINT "Numar      caracter    hexazecimal      octal"
60    PRINT N,CHR$(N),HEX$(N),OCT$(N)
```

```
70    INPUT "Continuati ? (Y/N) : ",C$
80   WEND
90   END
MERGE "C7"
Ok
RUN
Conversie numar -> caracter
Nr.( 32 <= N <= 126 ) = 66
Numar      caracter     hexazecimal        octal
 66           B             42              102
Continuati ? (Y/N) : Y
Nr.( 32 <= N <= 126 ) = 125
Numar      caracter     hexazecimal        octal
 125          }             7D              175
Continuati ? (Y/N) : N
Ok
```

Listing 5.35

În afară de valoarea numerică introdusă şi caracterul ASCII corespunzător acesteia, se afişează şi transcrierea hexazecimală şi octală corespunzătoare, în scopul facilitării înţelegerii celor două interpretări ale valorii binare memorate într-un octet.

De exemplu:

$$66_{(10)} = 102_{(8)} = 42_{(16)} = 1000010_{(2)}$$

dar configuraţia binară 01000010 (anexa B) corespunde codului ASCII al caracterului B. Deci interpretată ca valoare numerică secvenţa binară 01000010 reprezintă numărul zecimal 66, pe cînd caracterul reprezintă litera B.

● Crearea unui cap de tabel

Aplicaţia este prezentată în listingul 5.36.

```
10    PRINT "Creare cap tabel"
20    M$="M  A  T  E  R  I  I"
30    E$="= Numele si prenumele elevului |"
40    A$="|  romana  |matematica|  fizica  |  chimie  ="
50    CR$=CHR$(13)+CHR$(10)
60    PRINT STRING$(76,61);CR$;"=";SPC(30);"|";SPC(12);
      M$;SPC(12);"=";CR$;E$;STRING$(43,45);"=";CR$;
      "=";SPC(30);A$;CR$;"=";STRING$(74,45);"="
70    END
RUN
Creare cap tabel
============================================================================
=                              |            M  A  T  E  R  I  I              =
= Numele si prenumele elevului |--------------------------------------------=
=                              |  romana  |matematica|  fizica  |  chimie  =
============================================================================
Ok
```

Listing 5.36

● Conversia unui şir de cifre în valoare numerică

Aplicaţia, prezentată în listingul 5.37, verifică şirul introdus caracter cu caracter şi, dacă este corect, efectuează conversia acestuia în număr zecimal. Conversia se efectuează cu funcţia VAL (linia 110).

```
10    PRINT "Validare si conversie"
20    PRINT "sir de cifre -> valoare numerica"
30    C$="Y"
40    WHILE C$="Y" OR C$="y"
50    INPUT "NR :",N$
60    BUN=1
70    FOR I=1 TO LEN(N$)
80    A$=MID$(N$,I,1)
90    IF NOT (A$=>"0" AND A$<="9") AND A$<>"."
          THEN BUN=0
100       NEXT
110       IF BUN<>1
            THEN PRINT "Numar incorect"
            ELSE N=VAL(N$):
           PRINT "S-a tastat numarul :";N
120       INPUT "Continuati ? (Y/N) : ",C$
130   WEND
140   END
RUN
Validare si conversie
sir  de cifre -> valoare numerica
NR :12.3
S-a tastat numarul : 12.3
Continuati ? (Y/N) : Y
NR :17P8
Numar incorect
Continuati ? (Y/N) : Y
NR :1708
S-a tastat numarul : 1708
Continuati ? (Y/N) : N
Ok
```

Listing 5.37

● Căutarea unui subşir într-un şir dat

Aplicaţia este prezentată în listingul 5.38.

```
10    PRINT "Cauta si localizeaza un subsir"
20    PRINT "intr-un sir dat"
30    C$="Y"
40    WHILE C$="Y" OR C$="y"
50       PRINT "Introduceti sirul :"
60       INPUT "",A$
70       INPUT "Subsir :",B$
80       WHILE B$<>""
90    N=INSTR(1,A$,B$)
100   IF N=0
         THEN PRINT "Subsir inexistent"
         ELSE PRINT "Subsirul incepe la pozitia";N
```

```
110    INPUT "Subsir :",B$
120       WEND
130       INPUT "Continuati ? (Y/N) : ",C$
140    WEND
150    END
RUN
Cauta si localizeaza un subsir
intr-un sir dat
Introduceti sirul :
Acesta este sirul dat
Subsir :SIR
Subsir inexistent
Subsir :sir
Subsirul incepe la pozitia 13
Subsir :
Continuati ? (Y/N) : N
Ok
```

Listing 5.38

5.6 Fişiere

În capitolul 3, exemplul 3.19 a fost creat fişierul secvenţial CLASA. Un program pentru listarea acestuia s-a prezentat în exemplul 3.20, fiind ulterior salvat în fişierul LIST. De asemenea a fost ilustrat modul de creare a unui fişier în acces direct (fişierul CLASAD - exemplul 3.21).

Subcapitolul de faţă prezintă două aplicaţii ce exemplifică modurile de actualizare ale fişierelor CLASA (aplicaţia 39) şi CLASAD (aplicaţia 40).

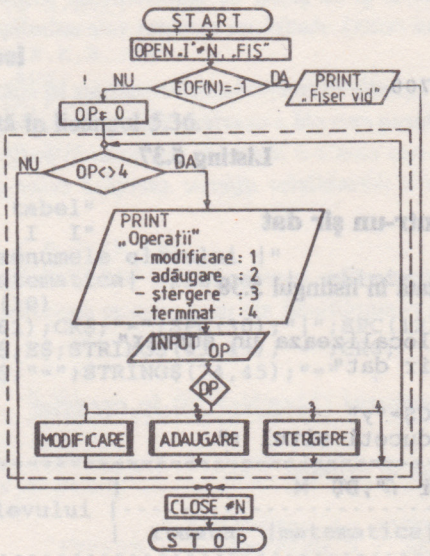


Fig. 5.26

Deoarece programele de actualizare a fişierelor sînt în general destul de complexe şi uneori pun probleme, prezentăm în figura 5.26 o schemă logică principială de abordare a unor astfel de probleme.

După cum se poate observa, dacă deschiderea fişierului s-a efectuat cu succes, utilizatorul se află în faţa unui **meniu** şi, funcţie de opţiunea OP, poate alege procedura de prelucrare dorită. La rîndul lor procedurile pot fi împărţite în module, astfel încît să se obţină o parcurgere arborescentă, clară şi simplă a tuturor posibilităţilor (necesităţilor) de prelucrare. După parcurgerea procedurii alese se revine la **meniul principal**.

Remarcă. În lucrul cu fişiere în acces direct gestiunea articolelor (a numerelor înregistrărilor) scrise, şterse sau neutilizate rămîne în sarcina utilizatorului. Din această cauză actualizarea fişierelor în acces direct trebuie executată cu multă atenţie deoarece există pericolul (în cazul utilizării neatente a numerelor înregistrărilor) de distrugere a unor informaţii utile.

Observaţie. Ambele programe prezentate în aplicaţiile 39 şi 40 au fost create şi testate în BASIC-80 (MBASIC,GBASIC) sub CP/M.

● Actualizarea fişierului secvenţial CLASA

Aplicaţia este prezentată în listingul 5.39.

```
10    PRINT "Actualizarea fisierului secvential CLASA"
20    C.MIN=1 : C.MAX=10
30    OPEN "I",#1,"CLASA"
40    IF EOF(1)=-1
      THENPRINT "Fisierul clasa este vid":
      ER%=-15
      ELSEOP=0:
      WHILE OP<>4:
          GOSUB 80:
      WEND
50    IF ER%=0
      THEN PRINT "Actualizare Ok"
      ELSE PRINT "ER =";ER%
60    CLOSE #1
70    END
80    'Precizare operatie
90    PRINT "Operatii :"
100   PRINT " - modificare (1)"
110   PRINT " - adaugare   (2)"
120   PRINT " - stergere   (3)"
130   PRINT " - terminare  (4)"
140   INPUT "Operatie (1/2/3/4) : ",OP
150   ON OP GOSUB 180,370,670
160   RETURN
170   'Modificare
180   INPUT "Nume     :",NUME$
190   WHILE NUME$<>""
200       INPUT "Prenume :",PREN$
210       ELEV$=NUME$+" "+PREN$
220       OPEN "O",#2,"MAN"
```

```
230         IND=0
240         WHILE EOF(1)<>-1
250     INPUT #1,E$,MR,MM,MF,MC
260     IF E$<>ELEV$
           THEN WRITE #2,E$,MR,MM,MF,MC
           ELSE GOSUB 870:
        IND=1
270          WEND
280          IF IND<>1
                THEN PRINT "Elev inexistent"
290         CLOSE #1
300         CLOSE #2
310         KILL "CLASA"
320         NAME "MAN" AS "CLASA"
330         OPEN "I",#1,"CLASA"
340         INPUT "Nume     :",NUME$
350     WEND
360     RETURN
370     'Adaugare
380     OPEN "O",#2,"MAN"
390     WHILE EOF(1)<>-1
400         INPUT #1,E$,MR,MM,MF,MC
410         WRITE #2,E$,MR,MM,MF,MC
420     WEND
430     CLOSE #1
440     INPUT "Nume     :",NUME$
450     WHILE NUME$<>""
460         INPUT "Prenume :",PREN$
470         ELEV$=NUME$+" "+PREN$
480         C.MAT$="Romana     "
490         GOSUB 3180 : MR=C.A
500         C.MAT$="Matematica"
510         GOSUB 3180 : MM=C.A
520         C.MAT$="Fizica     "
530         GOSUB 3180 : MF=C.A
540         C.MAT$="Chimie     "
550         GOSUB 3180 : MC=C.A
560         PRINT ELEV$;" : ";MR;MM;MF;MC
570         INPUT "Corect ? (Y/N) : ",CD$
580         IF CD$="Y" OR CD$="y"
              THEN WRITE #2,ELEV$,MR,MM,MF,MC
590         INPUT "Nume     :",NUME$
600     WEND
610     CLOSE #2
620     KILL "CLASA"
630     NAME "MAN" AS "CLASA"
640     OPEN "I",#1,"CLASA"
650     PRINT "Adaugare Ok"
660     RETURN
670     'Stergere
680     INPUT "Nume     :",NUME$
690     WHILE NUME$<>""
700         INPUT "Prenume :",PREN$
710         ELEV$=NUME$+" "+PREN$
720         OPEN "O",#2,"MAN"
730         INDA=0
740         WHILE EOF(1)<>-1
750     INPUT #1,E$,MR,MM,MF,MC
760     IF E$<>ELEV$
           THEN WRITE #2,E$,MR,MM,MF,MC
```

```
        ELSE INDA=1:
    GOSUB 1040:
    INPUT "Stergeti ? (Y/N) : ",CD$:
    IF CD$="Y" OR CD$="y"
       THEN PRINT "Articol sters"
       ELSE WRITE #2,E$,MR,MM,MF,MC:
    PRINT "Articol salvat"
770     WEND
780     IF INDA<>1
        THEN PRINT "Elev inexistent"
790     CLOSE #1
800     CLOSE #2
810     KILL "CLASA"
820     NAME "MAN" AS "CLASA"
830     OPEN "I",#1,"CLASA"
840     INPUT "Nume     :",NUME$
850   WEND
860   RETURN
870    Modificare medii
880   INDA=0
890   WHILE INDA<>1
900     GOSUB 1040
910     C.MAT$="Romana     "
920     GOSUB 3180 : MR=C.A
930     C.MAT$="Matematica"
940     GOSUB 3180 : MM=C.A
950     C.MAT$="Fizica     "
960     GOSUB 3180 : MF=C.A
970     C.MAT$="Chimie     "
980     GOSUB 3180 : MC=C.A
990     PRINT ELEV$;" : ";MR;MM;MF;MC
1000    INPUT "Corect ? (Y/N) : ",CD$
1010    IF CD$="Y" OR CD$="y"
         THEN WRITE #2,ELEV$,MR,MM,MF,MC:
        PRINT "Modificare Ok":
        INDA=1
1020  WEND
1030  RETURN
1040  'Afisare elev
1050  PRINT "Elev : ",E$
1060  PRINT " romana   matematica   fizica   chimie"
1070  PRINT USING "  ##     ";MR;MM;MF;MC
1080  RETURN
SAVE "ACT",A
Ok
LOAD "LIST",R
Listarea fisierului secvential CLASA
Numele si prenumele elevului    romana  matematica     fizica     chimie
Alexe Ion                          7          8           9          9
Coman Lili                         6          7           6          7
Dobre Nicu                         7          8           8          9
Ok
LOAD "ACT",R
Actualizarea fisierului secvential CLASA
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 2
```

```
Nume     :Florea
Prenume :Gheorghe
Romana      ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica      ( 1 <= N <= 10 ) = 9
Chimie      ( 1 <= N <= 10 ) = 9
Florea Gheorghe :  9  10  9  9
Corect ? (Y/N) : Y
Nume     :Nicolau
Prenume :Carmen
Romana      ( 1 <= N <= 10 ) = 8
Matematica( 1 <= N <= 10 ) = 9
Fizica      ( 1 <= N <= 10 ) = 8
Chimie      ( 1 <= N <= 10 ) = 8
Florea Gheorghe :  8  9  8  8
Corect ? (Y/N) : Y
Nume     :
Adaugare Ok
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 1
Nume     :Coman
Prenume :Lili
Elev :          Coman Lili
 romana    matematica    fizica    chimie
    6           7           6         7
Romana      ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica      ( 1 <= N <= 10 ) = 10
Chimie      ( 1 <= N <= 10 ) = 9
Coman Lili :  9  10  10  9
Corect ? (Y/N) : Y
Modificare Ok
Nume     :
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 3
Nume     :Dobre
Prenume :Nicu
Elev :          Dobre Nicu
 romana    matematica    fizica    chimie
    7           8           8         9
Stergeti ? (Y/N) : Y
Articol sters
Nume     :
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 4
Actualizare Ok
Ok
LOAD "LIST",R
```

```
Listarea fisierului secvential CLASA
Numele si prenumele elevului     romana  matematica     fizica     chimie
Alexe Ion                             7           8          9          9
Coman Lili                            9          10         10          9
Florea Gheorghe                       9          10          9          9
Nicolau Carmen                        8           9          8          8
Ok
```

Listing 5.39

● Actualizarea fişierului cu acces direct CLASAD

Aplicaţia este prezentată în listingul 5.40.

```
10    PRINT "Actualizarea fisierului"
20    PRINT "cu acces direct CLASAD"
30    C.MIN=1 : C.MAX=10
40    OPEN "R",#1,"CLASAD",40
50    FIELD #1,28 AS EL$,3 AS MR$,3 AS MM$,3 AS MF$,3 AS MC$
60    OP=0
70    WHILE OP<>4
80        PRINT "Operatii :"
90        PRINT " - modificare (1)"
100       PRINT " - inserare   (2)"
110       PRINT " - stergere   (3)"
120       PRINT " - terminare  (4)"
130       INPUT "Operatie (1/2/3/4) : ",OP
140       ON OP GOSUB 180,280,360
150   WEND
160   PRINT "Actualizare Ok"
170   END
180   'Modificare
190   INPUT "Numar inregistrare : ",NR%
200   WHILE NR%>0
210       GET #1,NR%
220       GOSUB 550
230       GOSUB 590
240       INPUT "Numar inregistrare : ",NR%
250   WEND
260   PRINT "Modificare Ok"
270   RETURN
280   'Inserare
290   INPUT "Numar inregistrare : ",NR%
300   WHILE NR%>0
310       GOSUB 630
320       INPUT "Numar inregistrare : ",NR%
330   WEND
340   PRINT "Inserare Ok"
350   RETURN
360   'Stergere
370   INPUT "Numar inregistrare : ",NR%
380   WHILE NR%>0
390       GET #1,NR%
400       GOSUB 550
410       INPUT "Stergeti articolul ? (Y/N) : ",CD$
420       IF CD$="Y" OR CD$="y"
            THEN GOSUB 470:
           PRINT "Articol sters"
```

```
          ELSE PRINT "Articol salvat"
430     INPUT "Numar inregistrare : ",NR%
440   WEND
450   PRINT "Stergere Ok"
460   RETURN
470   '
480   LSET EL$=SPACE$(28)
490   LSET MR$=SPACE$(3)
500   LSET MF$=SPACE$(3)
510   LSET MM$=SPACE$(3)
520   LSET MC$=SPACE$(3)
530   PUT #1,NR%
540   RETURN
550   'Afisare
560   PRINT "Elev : ";EL$
570   PRINT "romana:";MR$;"    matematica:";MM$;
      "    fizica:";MF$;"    chimie:";MC$
580   RETURN
590   '
600   INPUT "Modificati articolul ? (Y/N) : ",CD$
610   IF CD$="Y" OR CD$="y"
        THEN GOSUB 630
620   RETURN
630   'Introducere articol
640       INPUT "Nume     :",NUME$
650       INPUT "Prenume  :",PREN$
660       ELEV$=NUME$+" "+PREN$
670       C.MAT$="Romana    "
680       GOSUB 3180 : MR=C.A
690       C.MAT$="Matematica"
700       GOSUB 3180 : MM=C.A
710       C.MAT$="Fizica    "
720       GOSUB 3180 : MF=C.A
730       C.MAT$="Chimie    "
740       GOSUB 3180 : MC=C.A
750       PRINT ELEV$;" : ";MR;MM;MF;MC
760       INPUT "Articol corect ? (Y/N) : ",C$
770       IF C$="Y" OR C$="y"
            THEN LSET EL$=ELEV$ :
           RSET MR$=STR$(MR) :
           RSET MM$=STR$(MM) :
           RSET MF$=STR$(MF) :
           RSET MC$=STR$(MC) :
           PUT #1,NR%
780   RETURN
MERGE "C7"
Ok
RUN
Actualizarea fisierului
cu acces direct CLASAD
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 1
Numar inregistrare : 3
Elev : Luca Ion
romana:  9    matematica: 10    fizica:  9    chimie:  9
Modificati articolul ? (Y/N) : Y
```

```
Nume     :Luca
Prenume :Ion
Romana     ( 1 <= N <= 10 ) = 9
Matematica( 1 <= N <= 10 ) = 10
Fizica     ( 1 <= N <= 10 ) = 9
Chimie     ( 1 <= N <= 10 ) = 8
Florea Gheorghe :  9  10  9  8
Articol  corect ? (Y/N) : Y
Numar inregistrare :
Modificare Ok
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 2
Numar inregistrare : 4
Nume     :Rusu
Prenume :Emil
Romana     ( 1 <= N <= 10 ) = 8
Matematica( 1 <= N <= 10 ) = 7
Fizica     ( 1 <= N <= 10 ) = 7
Chimie     ( 1 <= N <= 10 ) = 7
Florea Gheorghe :  8  7  7  7
Articol  corect ? (Y/N) : Y
Numar inregistrare :
Inserare Ok
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 3
Numar inregistrare : 5
Elev : Toader Virgil
romana:  9    matematica: 10    fizica:  8    chimie:  8
Stergeti articolul ? (Y/N) : Y
Articol sters
Numar inregistrare :
Stergere Ok
Operatii :
 - modificare (1)
 - adaugare   (2)
 - stergere   (3)
 - terminare  (4)
Operatie (1/2/3/4) : 4
Actualizare Ok
Ok
```

Listing 5.40

ANEXA A

PRINCIPALELE CARACTERISTICI ALE MODULELOR BASIC UTILIZATE ÎN EXEMPLELE PREZENTATE

Fişier rutină	Numere linii	Funcţia realizată	Variabile			ER%[1]	Ape-luri	Lstg. defi-nire
			Intrare	Ieşire	Globale			
R1	1000-1010	Împărţirea cu rest (N2<>0)			N2,N1	-1		5.1
R2	1020-1060	CMMDC al două numere naturale diferite de zero	R.A	R.C		-2		4.9
R3	1065-1096	CMMDC al două numere întregi	R.A R.B	R.C		-3 -4		5.3
R4	1100-1115	CMMDC al două numere întregi	R.A R.B	R.C		-3 -4	R3	5.4
R5	1120-1135	CMMDC al R.N numere întregi (2≤R.N≤20)	R.N	R.C	A(20)	-5	R3	5.5
R7	1180-1280	CMMDC al R.N numere întregi (2≤R.N≤20)	R.N	R.C	A(20)	-5	R3	5.6
R8	1300-1375	Determinarea numerelor prime ≤ R.N (2≤R.N≤1000)	R.N		PRIM(300)	-6		5.7
R9	1380-1420	Converteşte R.N din baza 10 în baza R.B (0≤R.N≤32767; 2≤R.B≤9)	R.N R.B		CIFRE(16) ORD	-3		5.9
R10	1440-1470	Converteşte în baza 10 un număr de 16 cifre binare	R.A		CIFRE(16)	-8 -9		5.11
R11	1480-1565	Descompunerea unui număr întreg în factori primi	R.N	R.NR	F(100) P(100)	-3		5.12
R12	1590-1655	Determină minimul (R.I=0) sau maximul (R.I<>0) şi locul său în V(); (1≤R.N≤20)	R.N R.I	R.M R.L	V(20)	-2		5.13

R13	1660-1675	Suma elementelor vectorului V() (1≤R.N≤20)	R.N	R.S	V(20)	-2		5.14
R14	1680-1730	Caută în V() elementul R.NR (1≤R.N≤20)	R.NR	R.L	V(20)	-2		5.15
R15	1740-1810	Detremină numărul elementelor negative nule şi pozitive din V(); (1≤R.N≤20)	R.N	R.NG R.Z R.P	V(20)	-2		5.16
R16	1820-1830	Schimbă poziţiile a două elemente ale lui V() între ele (1≤R.N≤20)	R.N R.A R.B		V(20)	-2		5.17
R17	1840-1850	Inversează ordinea elementelor lui V(); (1≤R.N≤20)	R.N		V(20)	-2		5.18
R18	1860-1930	Ordonarea crescătoare / descrescătoare a elementelor lui V() (2≤R.N≤20)	R.N R.I		V(20)	-5		5.20
R19	1940-1950	Mută R.K elemente de la începutul la sfirşitul lui V() (2≤R.N≤20)	R.N R.K		V(20)	-2 -5		5.19
R20	1960-2045	Rotirea stînga (R.I>0) sau dreapta (R.I<0) a elementelor lui V(); (2≤R.N≤20)	R.N R.I		V(20)	-2 -5		5.21
R21	2050-2125	Deplasarea stînga (R.I>0) sau dreapta (R.I<0) a elementelor lui V(); (2≤R.N≤20)	R.N R.I		V(20)	-2 -5		5.22
R22	2130-2170	Verifică dacă V() este o mulţime (1≤R.N≤10)	R.N		V(10)	-10 -11		5.23
R23	2175-2220	Reuniunea mulţimilor A() şi B() (1≤R.N,R.M≤10)	R.N R.M	R.K	A(10) B(10) C(20)	-10 -11		5.24
R24	2230-2265	Intersecţia mulţimilor A() şi B() (1≤R.N,R.M≤10)	R.N R.M	R.K	A(10) B(10) C(10)	-10 -11		5.25

R25	2280-2325	Diferenţa mulţimilor A() \ B() (1≤R.N,R.M≤10)	R.N R.M	R.K	A(10) B(10) C(20)	-10 -11		5.26
R26	2330-2450	Calculul valorii unui determinant (1≤R.N≤10; R.EPS≥0.0001)	R.EPS R.N	R.D	M(10,10)	-12 -13		5.33
R27	2460-2615	Inversa unei matrice (1≤R.N≤10; R.EPS≥0.0001)	R.N R.EPS		M(10,10)	-12 -13 -14		5.34
C1	3000-3030	Citeşte şi validează N1, N2 N*			N1, N2			4.9
C2	3040-3055	Citeşte un număr întreg		C.A				5.2
C3	3060-3095	Citeşte un vector de maxim 20 de numere întregi			N, A(20)			5.5
C4	3100-3130	Citeşte un număr binar de maxim 16 cifre (1≤C.A-≤16)	C.A		CIFRE (16)	-6		5.8
C5	3135-3150	Citeşte şi validează un număr natural diferit de zero		C.A				5.13
C6	3155-3175	Citeşte un şir de C.A numere în V(); (1≤C.A≤20)	C.A		V(20)	-2		5.13
C7	3180-3200	Citeşte şi validează un număr întreg cuprins între C.MIN şi C.MAX	C.MIN C.MAX C.MAT$	C.A				3.19
C8	3210-3260	Citeşte şi validează o mulţime de R.N elemente		R.N	V(10)		C(7) R18 R22	5.23
C9	3270-3280	Citeşte o matrice de C.A x C.B elemente (1≤C.A,C.B≤10)	C.A C.B		M(10,10)	-12		5.28
S1	5000-5010	Afişează CMMDC			CMMDC			4.6
S2	5020-5030	Afişează CMMMC			CMMMC			4.9

S3	5040-5090	Determină şi afişează divizorii unui număr întreg	R.N					5.8
S4	5100-5145	Afişează un vector de S.N valori numerice (1≤S.N≤20)	S.N S.V$		V(20)			5.17
S5	5150-5195	Afişează componentele unei mulţimi V(); (1≤S.N≤20)	S.N S.V$		V(20)			5.24
S6	5200-5210	Afişează elementele matricei M() (1≤S.A,S.B≤10)	S.A S.B		M(10,10)	-12		5.28

[1] Coduri de eroare returnate de module (ER%):

0	execuţie normală (fără erori)
-1	împărţire la zero
-2	N1 sau N2 în afara limitelor
-3	N1 sau N2 sînt numere întregi
-4	în determinarea lui CMMDC, toate numerele sînt nule
-5	parametru mai mic decît 2 sau mai mare decît 20
-6	parametru mai mic decît 2 sau mai mare decît 1000
-7	baza mai mică decît 2 sau mai mare decît 9
-8	ORD<0 sau ORD>15
-9	într-un număr binar s-au detectat cifre diferite mai mari decît 1
-10	şir neordonat sau element dublu definit într-o mulţime
-11	parametru mai mic decît 1 sau mai mare decît 10
-12	parametru în afara limitelor în R26 sau R27
-13	$\|R.EPS\|<10^{-4}$
-14	matrice singulară
-15	fişier vid

ANEXA B

Codul ASCII

Zecimal	Binar	Octal	Hexa	Caracter
000	0000000	000	00	<NUL>
001	0000001	001	01	<SOH> (Start of heading)
002	0000010	002	02	<STX> (Start of text)
003	0000011	003	03	<ETX> (End of text)
004	0000100	004	04	<END> (End of transmission)
005	0000101	005	05	<ENQ> (Enquiry)
006	0000110	006	06	<ACK> (Acknowledge)
007	0000111	007	07	<BELL>
008	0001000	010	08	<BS> (Backspace)
009	0001001	011	09	<HT> (Horizontal tabulation)
010	0001010	012	0A	<LF> (Line feed)
011	0001011	013	0B	<VT> (Vertical tabulation)
012	0001100	014	0C	<FF> (Form feed)
013	0001101	015	0D	<CR> (Carriage return)
014	0001110	016	0E	<SO> (Shift out)
015	0001111	017	0F	<SI> (Shift in)
016	0010000	020	10	<DLE> (Data link espace)
017	0010001	021	11	<DC1> (Device control 1)
018	0010010	022	12	<DC2> (Device control 2)
019	0010011	023	13	<DC3> (Device control 3)
020	0010100	024	14	<DC4> (Device control 4)
021	0010101	025	15	<NAK> (Negative aknowledge)
022	0010110	026	16	<SYN> (Synchronous idle)

023	0010111	027	17	<ETB> (End of transmission block)
024	0011000	028	18	<CAN> (Cancel)
025	0011001	031	19	<EM> (End of medium)
026	0011010	032	1A	<SUB> (Substitute)
027	0011011	033	1B	<ESCAPE>
028	0011100	034	1C	<FS> (File separator)
029	0011101	035	1D	<GS> (Group separator)
030	0011110	036	1E	<RS> (Record separator)
031	0011111	037	1F	<US> (Unit separator)
032	0100000	040	20	<SPACE>
033	0100001	041	21	!
034	0100010	042	22	"
035	0100011	043	23	#
036	0100100	044	24	$
037	0100101	045	25	%
038	0100110	046	26	&
039	0100111	047	27	'
040	0101000	050	28	(
041	0101001	51	29	)
042	0101010	52	2A	*
043	0101011	53	2B	+
044	0101100	054	2C	,
045	0101101	055	2D	-
046	0101110	056	2E	.
047	0101111	057	2F	/
048	0110000	060	30	0
049	0110001	061	31	1
050	0110010	062	32	2

051	0110011	063	33	3
052	0110100	064	34	4
053	0110101	065	35	5
054	0110110	066	36	6
055	0110111	067	37	7
056	0111000	070	38	8
057	0111001	071	39	9
058	0111010	072	3A	:
059	0111011	073	3B	;
060	0111100	074	3C	<
061	0111101	075	3D	=
062	0111110	076	3E	>
063	0111111	077	3F	?
064	1000000	100	40	@
065	1000001	101	41	A
066	1000010	102	42	B
067	1000011	103	43	C
068	1000100	104	44	D
069	1000101	105	45	E
070	1000110	106	46	F
071	1000111	107	47	G
072	1001000	110	48	H
073	1001001	111	49	I
074	1001010	112	4A	J
075	1001011	113	4B	K
076	1001100	114	4C	L
077	1001101	115	4D	M
078	1001110	116	4E	N

079	1001111	117	4F	O
080	1010000	120	50	P
081	1010001	121	51	Q
082	1010010	122	52	R
083	1010011	123	53	S
084	1010100	124	54	T
085	1010101	125	55	U
086	1010110	126	56	V
087	1010111	127	57	W
088	1011000	130	58	X
089	1011001	131	59	Y
090	1011010	132	5A	Z
091	1011011	133	5B	[
092	1011100	134	5C	\
093	1011101	135	5D	]
094	1011110	136	5E	^
095	1011111	137	5F	_
096	1100000	140	60	‘
097	1100001	141	61	a
098	1100010	142	62	b
099	1100011	143	63	c
100	1100100	144	64	d
101	1100101	145	65	e
102	1100110	146	66	f
103	1100111	147	67	g
104	1101000	150	68	h
105	1101001	151	69	i
106	1101010	152	6A	j

107	1101011	153	6B	k
108	1101100	154	6C	l
109	1101101	155	6D	m
110	1101110	156	6E	n
111	1101111	157	6F	o
112	1110000	160	70	p
113	1110001	161	71	q
114	1110010	162	72	r
115	1110011	163	73	s
116	1110100	164	74	t
117	1110101	165	75	u
118	1110110	166	76	v
119	1110111	167	77	w
120	1111000	170	78	x
121	1111001	171	79	y
122	1111010	172	7A	z
123	1111011	173	7B	{
124	1111100	174	7C	|
125	1111101	175	7D	}
126	1111110	176	7E	~
127	1111111	177	7F	<DEL> (Delete)

Bibliografie

[1] Dodescu Gh.,Ionescu D., Nisipeanu L., Pilat F.-'Limbajul BASIC şi aplicaţii', Editura Didactică şi Pedagogică, Bucureşti 1978

[2] Dumitraşcu L., Ioachim Al. - 'Tehnici de construire a programelor cu structuri alternative', Editura Academiei R.S.R, 1981

[3] Dumitraşcu L. - 'BASIC pentru începători cu calculatorul personal', A.M.C. Nr. 48, Editura Tehnică, Bucureşti 1986

[4] Dumitraşcu L. - 'Învăţăm microelectronica interactivă', Editura Tehnică, Bucureşti 1989

[5] Dobrescu P. - 'Computere şi trandafiri sau paradoxurile progresului', Editura Politică, Bucureşti 1988

[6] Iorgulescu A. - 'Exerciţii de programare structurată în FORTRAN', Atelierul poligrafic ASE, Bucureşti 1980

[7] Moldovan G. - 'Scheme logice şi programe FORTRAN', Editura Didactică şi Pedagogică, Bucureşti 1978

[8] Moraru F., Atodiroaei M. - 'Programarea microcalculatoarelor în sistemul de operare CP/M', Editura Ştiinţifică şi Enciclopedică, Bucureşti 1989

[9] Niculescu St., Dumitrescu M. - 'Algoritmi şi metode de reprezentare', Editura Tehnică, Bucureşti 1978

[10] Niculescu St. - 'FORTRAN iniţiere în programarea structurată' Editura Tehnică, Bucureşti 1979

[11] Petrescu A. ş.a. - 'Totul despre ... calculatorul personal aMIC ', Editura Tehnică, Bucureşti 1984

[12] Petrescu A. ş.a. - 'Microcalculatoarele Felix M18, M118B, M118', Editura Tehnică, Bucureşti 1984

[13] Petrescu A. ş.a. - 'ABC de calculatoare personale şi ... nu doar atît ...', Editura Tehnică, Bucureşti 1990

[14] Văduva I., Baltac V., Florescu V., Floricica I., Jitaru M. - 'Ingineria programării', Editura Academiei R.S.R., 1985

[15] Zaharia M. - 'FORTRAN-77. Depanarea programelor', Editura Tehnică, Bucureşti 1991

[16] * * * - 'Dicţionar de informatică', Editura Ştiinţifică şi Enciclopedică, Bucureşti 1981

[17] * * * - 'Calculatoare electronice din generaţia a cincea', Editura Academiei R.S.R., Bucureşti 1985

[18] * * * - 'BETA BASIC' V3.0, Manual de utilizare, Betasoft 1985

Copyright © 1992, Editura Tehnică
Toate drepturile asupra acestei ediţii
sînt rezervate editurii

Adresa: Editura Tehnică
Piaţa Presei Libere, 1
33 Bucureşti, România
cod 79738

Redactor: **ing. Silvia Cândea**
Coperta: **grafician Simona Dumitrescu**
Tehnoredactare: **ing. Silvia Cândea**

Coli de tipar 12; Bun de tipar: 15.1.92
C.Z.: 681.3.06
ISBN: 973-31-0433-7

Tiparul executat sub comanda nr. [illegible]
la IMPRIMERIA "ARDEALUL" CLUJ

Lei 210,–

ISBN: 973-31-0433-7